Supporting Data Privacy in P2P Systems

Mohamed Jawad, Patricia Serrano-Alvarado, Patrick Valduriez

Supporting Data Privacy in P2P Systems

Mohamed Jawad[1], Patricia Serrano-Alvarado[1], and Patrick Valduriez[2]

[1] LINA, University of Nantes
France
Name.Lastname@univ-nantes.fr
[2] INRIA and LIRMM, University of Montpellier
France
Patrick.Valduriez@inria.fr

Abstract. Peer-to-Peer (P2P) systems have been very successful for large-scale data sharing. However, sharing sensitive data, like in online social networks, without appropriate access control, can have undesirable impact on data privacy. Data can be accessed by everyone (by potentially untrusted peers) and used for everything (e.g., for marketing or activities against the owner's preferences or ethics). Hippocratic databases (HDB) provide an effective solution to this problem, by integrating purpose-based access control for privacy protection. However, the use of HDB has been restricted to centralized systems. This chapter gives an overview of current solutions for supporting data privacy in P2P systems, and develops in more details a complete solution based on HDB.

Keywords: data privacy, P2P systems, DHT, Hippocratic databases, purpose-based access control, trust.

[3] Work partially funded by the DataRing project of the French ANR.

Table of Contents

1 Introduction

Data privacy is the right of individuals to determine for themselves when, how, and to what extent information about them is communicated to others [40]. It has been treated by many organizations and legislations that have defined well accepted principles. According to OECD[3], data privacy should consider: collection limitation, purpose specification, use limitation, data quality, security safeguards, openness, individual participation, and accountability. From these principles, we underline *purpose* specification which states that data owners should be able to specify the data access purposes for which their data will be collected, stored, and used.

With the advent of Online Social Networks (OLSN), data privacy has become a major concern. An OLSN is formed by people having something in common and connected by social relationships, such as friendship, hobbies, or coworking, in order to exchange information [11]. Many communities use OLSNs to share data in both professional and non-professional environments. Examples of professional OLSNs are Shanoir [4], designed for the neuroscience community to

[3] Organization for Economic Co-operation and Development. One of the world's largest and most reliable source of comparable statistics on economic and social data (`http://www.oecd.org/`).

[4] `www.shanoir.org/`

archive, share, search, and visualize neuroimaging data, or medscape [5], designed for the medical community to share medical experience and medical data. There are also non-professional OLSNs for average citizens and amateurs in different domains such as Carenity [6], designed for patients and their relatives to share medical information about them in order to help medical research. Another example is DIYbio [7], dedicated to make biology accessible for citizen scientists, amateur biologists, and biological engineers, who share research results. The most popular OLSN, Facebook, with hundreds millions of users, enables groups of friends to share all kinds of personal information among themselves.

Scalable data sharing among community members is critical for an OLSN system. Two main solutions have emerged for scalable data sharing: cloud computing and Peer-to-Peer (P2P). Cloud computing promises to provide virtually infinite computing resources (e.g., CPU, storage, network) that can be available to users with minimal management efforts [26]. Data are stored in data centers, typically very large clusters of servers, operated by infrastructure providers such as Amazon, Google, and IBM. Among others, cloud computing exhibits the following key characteristics: (a) elasticity, as additional resources can be allocated on the fly to handle increased demands, (b) ease of maintenance, which is managed by cloud providers, and (c) reliability, thanks to multiple redundant sites. These assets make cloud computing suitable for OLSNs.

However, cloud computing proposes a form of centralized storage that implies many effects on sensitive data: (a) users need to trust the providers and their servers; (b) providers can use community information to make profits (e.g., profiling, marketing, advertising); and (c) users may find their data censored by providers. In particular, (a) is very hard to enforce as cloud providers can outsource data storage to other providers, yielding a chain of subcontractors (typically in different countries, each with a different legislation on data privacy) which is difficult to track.

As an alternative to a centralized data sharing solution, like the cloud, P2P provides a fully decentralized infrastructure. Examples of very popular P2P applications can be found in networking (e.g., Skype), search engines (e.g., YaCy), OLSN (e.g., Diaspora), and content sharing (e.g., BitTorrent). For instance, one third of the Internet traffic today is based on BitTorrent. P2P systems for data-centered applications offer valuable characteristics: (a) decentralized storage and control, so there is no need to trust one particular server; (b) data availability and fault tolerance, thanks to data replication; (c) scalability to store large amounts of data and manage high numbers of users; (d) autonomy, as peers can join and leave the network at will.

We claim that a P2P solution is the right solution to support the collaborative nature of OLSN applications as it provides scalability, dynamicity, autonomy, and decentralized control. Peers can be the participants or organizations in-

[5] `http://www.medscape.com/connect/`
[6] `http://www.carenity.com`
[7] `http://diybio.org`

volved in collaboration and may share data and applications while keeping full control over their (local) data sources.

However, despite their assets, P2P systems offer limited guarantees concerning data privacy. They can be considered as hostile because data, that can be sensitive or confidential, can be accessed by everyone (by potentially untrusted peers) and used for everything (e.g., for marketing, profiling, fraudulence, or for activities against the owner's preferences or ethics). Several P2P systems propose mechanisms to ensure privacy such as OceanStore [22], Past [33], and Freenet [8]. However, these solutions remain insufficient. Data privacy laws have raised the respect of user privacy preferences where purpose-based access is cornerstone. Managing data sharing, with trustworthy peers, for specific purposes and operations, is not possible in current P2P systems without adding new services.

Inspired by the Hippocratic oath and its tenet of preserving privacy, Hippocratic databases (HDB) [2] have incorporated purpose-based privacy protection, which allows users to specify the purpose for which their data are accessed. However, HDB have been proposed for centralized relational database systems.

Applied to P2P systems, HDBs could bring strong privacy support as in the following scenario. Consider an OLSN where patients share their own medical records with doctors and scientists, and scientists share their research results with patients and doctors. Scientists have access to patient medical records if their access purpose is for research on a particular disease. Doctors have access to research results for giving medical treatment to their patients. In this context, Hippocratic P2P data sharing can be useful. Producing new P2P services that prevent peers from disclosing, accessing, or damaging sensitive data, encourages patients (resp. scientists) to share their medical records (resp. results) according to their privacy preferences. Thus, the challenge is to propose services to store and share sensitive data in P2P systems taking into account access purposes.

This chapter is organized as follows. Section 2 surveys data privacy in P2P systems and gives a comparative analysis of existing solutions. Section 3 surveys HDBs. Section 4 introduces PriMod, a privacy model that applies HDB principles to data sharing in P2P systems. Section 5 describes PriServ, a privacy service that supports PriMod in structured P2P networks. Section 6 presents the PriServ prototype. Section 7 concludes.

2 Data privacy in P2P systems

P2P systems operate on application-level networks referred to as overlay networks. The degree of centralization and the topology of overlay networks have significant influence on properties such as performance, scalability, and security. P2P networks are generally classified into two main categories: pure and hybrid [29]. In pure P2P networks, all peers are equal and they can be divided in structured and unstructured overlays. In hybrid P2P networks (also called super-peer P2P networks), some peers act as dedicated servers for some other peers and have particular tasks to perform.

Unstructured P2P overlays are created in an ad-hoc fashion (peers can join the network by attaching themselves to any peer) and data placement is completely unrelated to their organization. Each peer knows its neighbors, but does not know the resources they have. Many popular applications operate as unstructured networks like Napster, Gnutella, Kazaa, and Freenet [8]. In those systems, content is shared among peers without needing to download it from a centralized server. Those systems vary, among others, in the way data is indexed, that implies the way data is searched. There exist two alternatives for indexing: centralized and distributed. In centralized indexes, a peer is responsible for managing the index of the system. This centralization facilitates data searching because requesting peers consult the central peer to obtain the location of the data, and then directly contact the peer where the data is located. Napster is an example of a system that maintains this type of index. In the distributed approach, each peer maintains part of the index, generally the one concerning the data they hold. Data searching is typically done by *flooding*, where requesting peers send the request to all of its neighbors which forward the request to all of their neighbors if they do not have the requested data, and so on. For large networks, a Time to Live (TTL) is defined to avoid contacting all peers at every request. Gnutella is a system that maintains this kind of index.

Structured P2P systems have emerged to improve the performance of data searching by introducing a particular structure into the P2P network. They achieve this goal by controlling the overlay topology, the content placement, and the message routing. Initial research on P2P systems led to solutions based on Distributed Hash Tables (DHTs) where a hash function maps a *key* to each peer (such key is considered as the peer's identifier). Data placement is based on mapping a *key* to each data item and storing the $(key, data)$ pair at the peer which identifier is equal or follows the key. Thus, the distributed lookup protocol supported by these systems efficiently locates the peer that stores a particular data item in $O(logN)$ messages. Representative examples of DHTs are Chord [36], Pastry [34], and Tapestry [41]. The reader can find an analysis of data sharing in DHT-based systems in [32].

Hybrid P2P networks contain a subset of peers (called super-peers) that provide services to some other peers. These services can be data indexing, query processing, access control, meta-data management, etc. With only one super-peer, the network architecture reduces to client-server. The organization of super-peers follows a P2P approach and they can communicate with each other in sophisticated manners. An example of these systems is Edutella [28].

Peers may be participants that share data, request information, or simply contribute to the storage system. Considering that there are peers that own data and do not necessarily act as servers of those data, we distinguish between three kinds of peers:

- **Requester.** A peer that requests data.
- **Server.** A peer that stores and provides data.
- **Owner.** A peer that owns and shares data.

This section surveys the P2P systems that deal with data privacy. Depending on their main application, the data privacy issues are different. We divide these systems into two main classes: those focusing on distributed data storage (Section 2.1) and those focusing on massive data sharing (Section 2.2). We compare them, in Section 2.3, based on their privacy protection guarantees and the techniques they use.

2.1 Privacy in distributed data storage systems

Distributed data storage systems are mainly used by users who want to benefit from large storage space and possibly share data with some other users. Usually, the users of these systems require the following privacy guarantees:

- Data storage: data are available for owners.
- Data protection against unauthorized reads: servers do not have the ability to read the data they store.
- Data protection against corruption and deletion: servers do not have the ability to corrupt or delete the data they store.
- A peer does not claim the property of data owned by another peer.

Various techniques and protocols should be employed together to ensure such guarantees. For instance, replication can be used to guarantee data availability, but to limit privacy breaches, data replicas must be stored at trusted servers. Thus, trust and access control techniques can be used with replication in order to ensure data privacy. In addition, data digital checksums and encryption can be used to protect ownership rights as well as data from unauthorized reads.

Past [33], OceanStore [22], and Mnemosyne [12] are examples of systems that use such techniques.

Past is a large-scale, Internet-based, global storage utility that provides scalability, high availability, persistence, and security. It relies on Pastry and uses smartcards, self certifying data, and certified-based trust in order to protect data content from malicious servers.

OceanStore is a cooperative infrastructure that provides a consistent, highly-available, durable, and secured storage utility. It relies on Tapestry and uses symmetric cryptography and access control techniques to protect data privacy from malicious peers.

Mnemosyne is a storage service that provides a high level of privacy by using a large amount of shared distributed storage to hide data. It relies on Tapestry and uses steganographic data, data whose presence among random data cannot be detected. This allows to protect data from malicious reads and suppressions.

2.2 Privacy in massive data sharing systems

Massive data sharing systems are mainly used by users who want to (a) share data in the system and (b) request and download data from the system. (b) is

probably the main reason why this type of system is so well-known and used, in particular, in multimedia file sharing. One important difference with data storage systems is that, in massive data sharing, data are massively duplicated. In this type of system, users search for privacy guarantees related to data and users.

1. The data privacy guarantees are:
   - Data storage: data are available to authorized owners and requesters.
   - Data protection against unauthorized reads: servers do not have the ability to read the data they store.
   - Data protection against corruption and deletion: servers do not have the ability to corrupt or delete the data they store.
   - A peer does not claim the property of data owned by another peer.
   - Limited disclosure: data are not provided to unauthorized requesters.
2. The user privacy guarantees (usually referred to as anonymity guarantees) are:
   - Users are not monitored in the system by other peers.
   - Users freedom of behavior is not limited by the system.
   - Users are protected against identity theft.

Data privacy guarantees in massive data sharing systems are similar to those of distributed data storage systems, the only difference is limited disclosure. As said before, data privacy can be protected by techniques such as access control, trust management, data encryption, and digital checksums. The difference here is the potential number of requesters. On the other hand, user privacy can be protected by using different anonymity techniques. However, ensuring user privacy may cause undesired effects on data privacy. Users want to protect their data privacy while remaining anonymous to behave freely, which increases the risk of violating the data privacy of others. This loop is probably the main reason we did not find in the literature systems that guarantee both data and user privacy.

Protecting data privacy. Data privacy in massive data sharing systems can be illustrated with the following systems.

Office SharePoint Workspace[8], previously known as Office Groove, is a desktop application designed for document collaboration within teams (i.e., workspaces). It is based on a partially centralized P2P system. Each user has a private editable copy of the workspace. Workspace copies are synchronized via the network in a P2P manner. Office SharePoint Workspace uses access control, trust, and encryption techniques in order to protect data privacy.

Piazza [38] is a data management system that enables sharing of XML documents in a distributed and scalable way. It is based on an unstructured P2P system. Although the goal and emphasis of Piazza is data sharing and not data privacy, the creators of Piazza proposed in [27] new techniques for publishing a single data instance in a protected form, thus enforcing data privacy.

[8] `http://office.microsoft.com/en-us/sharepoint-workspace`

OneSwarm [14] is a P2P service that provides users with explicit control over their data privacy by letting them determine how data are shared. It relies on BitTorrent and was designed to provide privacy-preserving data sharing. OneSwarm uses asymmetric cryptography, access control, trust, and communication anonymity in order to protect data privacy.

Other systems address the censorship problem which can have effects on data privacy since censorship can be a reason for data suppression. In some systems such as Usenet news[9], anyone who sees a message can post a cancel message to delete it, allowing censorship. Many systems have been proposed to resist to censorship. **Dagster** [37] is a censorship-resistant publishing scheme that intertwines legitimate and illegitimate data from web pages, so that a censor cannot remove *objectionable content* without simultaneously removing legally *protected content*. **Tangler** [39] is another censorship resistant publishing scheme based on the idea of intertwining data. Newly published documents are dependent on previous published blocks. This dependency, called entanglement, provides a user some incentive to replicate and store the blocks of other documents. Thus, data blocks are resistant to censorship and suppression.

Censorship-resistant schemes protect data only from suppression. Data privacy is not fully protected since any user can access these data.

Protecting user privacy. User privacy in massive data sharing systems can be illustrated with the following systems. They mostly use anonymity techniques to guarantee anonymous publishing and sharing.

Freenet [8] is a free P2P system that ensures anonymous file sharing, browsing, and publishing. It provides users with freedom of behavior by ensuring their anonymity. Freenet has its own key-based routing protocol (similar to that of a DHT), uses symmetric cryptography, and user and communication anonymity in order to guarantee user privacy.

SwarmScreen [7] is a privacy preserving layer for P2P systems that disrupts community identification by obfuscating users' network behavior. SwarmScreen relies on BitTorrent and was designed to provide user privacy through plausible deniability. Since a user behavior can be deduced by her interests, SwarmScreen connects the user to other users outside of her community of interest, which can disguise her interests and thus her behavior.

Many systems such as **ANts P2P**[10] and **MUTE**[11] have been proposed as anonymous P2P file sharing softwares. They use anonymity in order to make the user untrackable, hide her identity and encrypt everything she is sending/receiving from others.

Censorship-resistant systems such as **Dagster** and **Tangler** usually use anonymity techniques to hide users identities. Thus, it is not possible to enforce censorship on data belonging to a specific user.

[9] `http://usenet-news.net/`
[10] `http://antsp2p.sourceforge.net/`
[11] `http://mute-net.sourceforge.net`

2.3 Evaluation

In this section, we evaluate the P2P systems summarized in Table 1. We compare them in terms of privacy properties guaranteed and techniques used.

Application	P2P systems	Focus on	Relies on	Goals
Data storage	Past	Protecting data privacy	Pastry	Scalable, highly-available, persistent, and secure storage
	OceanStore		Tapestry	Consistent, highly-available, durable, and secured storage
	Mnemosyne		Tapestry	Steganographic data storage
Massive data sharing	Office SharePoint Workspace		Partially centralized P2P	Document sharing, team collaboration
	Piazza		Unstructured P2P	Scalable XML sharing, data management system
	OneSwarm		BitTorrent	Privacy preserving data sharing
	Dagster	Protecting data from censorship	Not specified	Censorship resistant data sharing
	Tangler		Own routing protocol	Censorship resistant data sharing
	Freenet	Protecting user privacy	Own routing protocol	Anonymous file sharing, freedom of speech
	SwarmScreen		BitTorrent	Privacy preserving data sharing

Table 1. Sample of P2P systems

Privacy properties. The data and user privacy properties we analyze are the following:

- Data protection against unauthorized reads.
- Data protection against corruption and deletion.
- Limited disclosure.
- Anonymity.
- Denial of linkability. Peers have the ability to deny the links they have with other peers.
- Content deniability. Peers have the ability to deny their knowledge on data content.

The two last properties are taken from [31].

Tables 2 and 3 provide a comparison of the privacy properties guaranteed by the P2P systems we evaluate. In these tables, a cell is kept blank when a privacy property is not guaranteed by the P2P system. *N/A* is used when information about a privacy property is not available in the literature.

P2P Systems	Privacy properties guaranteed			
	Protection against unauthorized reads	Protection against corruption and deletion	Limited disclosure	
			Owners	Requesters
Past		Yes, due to data digital checksums	Yes, due to access control (smartcards)	N/A
OceanStore	Yes, due to encryption	Yes, due to data digital checksums	Yes, due to access control	N/A
Mnemosyne	Yes, due to encryption			
Office SharePoint Workspace	Yes, due to encryption		Yes, due to access control	Yes, due to access control and trust techniques
Piazza	Yes, due to encryption		Yes, due to access control	Yes, due to access control
OneSwarm	Yes, due to encryption		Yes, due to access control	Yes, due to access control and trust techniques
Dagster	Yes, due to encryption			
Tangler	Yes, due to encryption			
Freenet	Yes, due to encryption	Yes, due to data digital checksums		
SwarmScreen				

Table 2. Comparison of P2P systems based on the privacy properties guaranteed

Data protection against unauthorized reads. In order to protect data from unauthorized reads, data encryption is used. Data encryption prevents server peers, and possibly malicious eavesdroppers and routing peers, from reading private data.

OceanStore, Mnemosyne, Office SharePoint Workspace, Piazza, OneSwarm, Dagster, Tangler, and **Freenet** use data encryption. In **Mnemosyne, Dagster, Tangler, Freenet**, and **SwarmScreen**, data are public. In **Past**, servers are not controlled and data are not encrypted, so data are not protected against unauthorized server reads.

Data protection against corruption and deletion. It is hard to prevent data from being corrupted or deleted. However, data integrity techniques and digital checksums can be used to help users to verify if data have suffered unauthorized changes.

OceanStore and **Freenet** use digital checksums to verify that data content has not been tampered with. **Past** uses smarcards to sign data in order to authenticate them and verifies if they have been modified or corrupted. Thus, although private data are not protected against corruption and deletion, malicious changes can be detected. Countermeasures can be taken against malicious peers in order to demotivate any future corruption. In other systems, data checksums are not used, so data changes cannot be detected.

P2P Systems	Privacy properties guaranteed, cont.				
	Anonymity			Denial of linkability	Content deniability
	Authors	Servers	Readers		
Past	Yes, due to Pseudonymity	Yes, due to Pseudonymity	Yes, due to Pseudonymity	Yes, due to Pseudonymity	
OceanStore					Yes, due to encryption
Mnemosyne					Yes, due to encryption
Office SharePoint Workspace	Yes, due to workspaces anonymity	Yes, due to workspaces anonymity	Yes, due to workspaces anonymity	Yes, due to workspaces anonymity	Yes, due to encryption
Piazza					Yes, due to encryption
OneSwarm	Only for third-party monitoring		Only for third-party monitoring	Yes, due to communication anonymity	Yes, due to encryption
Dagster	Yes, due to anonymous communication	Yes, due to anonymous communication	Yes, due to anonymous communication	Yes, due to anonymous communication	Yes, due to encryption
Tangler	Yes, due to anonymous identities	Yes, due to anonymous identities	Yes, due to anonymous identities	Yes, due to anonymous communication	Yes, due to encryption
Freenet	Yes, due to anonymous communication	Yes, due to anonymous identities	Yes, due to anonymous identities	Yes, due to anonymous communication	Yes, due to encryption
SwarmScreen	Yes, due to anonymous communication	Yes, due to anonymous communication	Yes, due to anonymous communication	Yes, due to anonymous communication	

Table 3. Comparison of P2P systems based on the privacy properties guaranteed, cont.

Limited disclosure. In order to limit data disclosure, access control is used to prevent unauthorized requesters from accessing data.

In addition to access control, encryption is used to prevent unauthorized disclosure due to collusion between servers and requesters. Even if an unauthorized requester receives encrypted data, it cannot access data content without having the corresponding decryption keys.

In addition, trust techniques can be used to make owners feel more comfortable about the use of their data as they can verify the trustworthiness of the requester.

Past and **OceanStore** use access control to limit disclosure only for authorized data owners.[12] **Freenet, Dagster**, and **Tangler** do not use access control since the peers are anonymous. **Piazza** uses access control to limit disclosure for authorized users. **Office SharePoint Workspace** and **OneSwarm** use access control but also trust techniques, thus, data disclosure is limited not only to authorized peers but also to trustworthy ones.

User anonymity. Four types of anonymity guarantees are defined in [10]:

1. Author (i.e., owner) anonymity: which users created which documents?

[12] These systems are not meant for massive data sharing, thus information about data disclosure for requesters is not available.

2. Server anonymity: which peers store a given document?
3. Reader (i.e., requester) anonymity: which users access which documents?
4. Document anonymity: which documents are stored at a given peer?

Past guarantees author and server anonymity due to pseudonymity techniques. Each user holds an initially unlinkable pseudonym in the form of a public key. The pseudonym is not easily linked to the user's real identity. If desired, a user may have several pseudonyms to hide that certain operations were initiated by the same user. **Past** users do not need to reveal their identity, nor the data they are retrieving, inserting, or storing.

Office SharePoint Workspace guarantees author, server, and reader anonymity, due to anonymous workspaces. Let us recall that inside workspaces, anonymity is not preserved.

OneSwarm and **SwarmScreen** guarantee author and reader anonymity due to anonymous communication but only from third-party monitoring. In OneSwarm, users' identities are known by servers in order to perform access control, so their anonymity is not guaranteed. Server anonymity is also not guaranteed because they must be easily located when publishing or requesting data.

Dagster and **Tangler** guarantee author, server, and reader anonymity, due to anonymous communication and anonymous identities. Because all connections between the server and the owner/requester are over an anonymous channel, there is no correlation between their identities and the documents they are publishing or requesting.

Freenet guarantees author, server, and reader anonymity due to anonymous communication. For reader and server anonymity, while a peer can get some indication on how early the request message is on the forwarding chain by using the limit on the number of hops (hop-to-live), the true reader and server are kept private due to anonymous communication. Author anonymity is protected by occasional resetting of the data source field in response messages. The peer appearing as the data source does not imply that it actually supplies that data.

Document anonymity is discussed later.

Denial of linkability. Linkability refers to identifying the correlation between users. Knowledge on linkability can be denied by using anonymous communication. Denial of linkability can protect peers from third-party monitoring.

Past uses pseudonymity techniques, thus knowledge on linkability may be denied by peers. **Office SharePoint Workspace** guarantees denial of linkability outside workspaces due to anonymous workspaces. **OneSwarm, Dagster, Tangler, Freenet**, and **SwarmScreen** guarantee denial of linkability, due to anonymous communication.

Content deniability. Content deniability refers to whether peers can deny the knowledge on the content stored or transmitted (document anonymity). Knowledge on content can be denied if the content is not readable by the peer that holds it.

In all systems that use data encryption, knowledge on data content can be denied. This is the case of **OceanStore, Mnemosyne, Office SharePoint Workspace, Piazza, OneSwarm, Dagster, Tangler**, and **Freenet**. In **Past** and **SwarmScreen**, servers can access data, so they cannot deny the knowledge on the data content they store.

Privacy techniques. The privacy techniques we analyze are the following:

- Access control. Data access can be controlled with respect to the identity of the user, her role, and the access purpose of the requester.
- Anonymity techniques. A user (resp. a data) is made indistinguishable from other users (resp. data), thus providing her anonymity among a group of users (resp. data set).
- Trust techniques. The behavior of users is predicted.
- Cryptography techniques. Data can be made "unreadable" by converting ordinary information (plain text) into unintelligible cipher text.

Table 4 provides a summary of the techniques used to ensure privacy by the systems we evaluate.

Access control. Access control is essential to guarantee that data will not be read or shared with unauthorized peers.

In **Past**, access control is based on the use of smartcards which generate and verify various certificates. Users may access data or not within the access rights related to their certificate.

In **OceanStore**, access control is based on two types of restrictions: *reader* and *writer* restrictions. In the *reader* restriction, to prevent unauthorized reads, the data decryption keys are distributed by the data owner to users with read permissions. To revoke read permissions, the data owner requests users to delete replicas or re-encrypt them with new encryption keys. A recently-revoked reader is able to read old data from cached copies or from misbehaving servers that fail to delete or re-encrypt. This problem is not specific to **OceanStore**, even in conventional systems, as there is no way to force readers to forget what has been read. To prevent unauthorized writes, they must be signed so that well-behaved servers and clients can verify them against an Access Control List (ACL). The data owner can define an ACL by datum by providing a signed certificate. ACLs are public so that servers can check whether a write is allowed. Thus, writes are restricted at servers by ignoring unauthorized updates.

In **Office SharePoint Workspace**, access control is based on the use of membership lists and workspace rules. Users are identified by accounts and passwords that allow them to log in workspaces. If they can log in a workspace W, they are listed in the membership list of W. A user can access or remove data from W as long as she is a member of W and, in addition, respects the workspace usage rules.

In **Piazza**, the access to a published XML document is restricted to parts of the document in accordance with the data owner preferences. Data owners

P2P Systems	Privacy techniques				
	Access control	Anonymity	Trust techniques	Cryptography techniques	Data integrity protection
Past	Use of smartcards	Pseudonymity using smarcards	Certificate-based Trust		Use of smartcards for data certification
OceanStore	Use of two types of restriction (reader and writer)			Symmetric encryption	Use of content hashing as a checksum
Mnemosyne				Symmetric block encryption	
Office SharePoint Workspace	Use of workspace and membership lists	Anonymous workspaces and groups	Use of trust colors	Symmetric encryption	
Piazza	Use of access policies defined by the owner			Symmetric encryption	
OneSwarm	Use of permissions defined by the owner	Anonymous communication	Use of community trust levels	Hybrid encryption	
Dagster		Anonymous communication		Symmetric block encryption	
Tangler		Anonymous communication and identities		Symmetric block encryption	
Freenet		Anonymous communication		Symmetric encryption	Use of signature-verifying keys
SwarmScreen		Random communication			

Table 4. Comparison of P2P systems based on used privacy techniques

in **Piazza** can specify access control policies declaratively and generate data instances that enforce them. By granting decryption keys to users, the data owner enforces an access control policy. Once published, the data owner relinquishes all control over who downloads and processes the data. Requesters can access the data conditionally, depending on the keys they possess.

In **OneSwarm**, persistent identities allow users to define per-file permissions. These permissions (i.e., capabilities) restrict access to protected data. For example, OneSwarm can be used to restrict the distribution of a photo file to friends and family only.

Anonymity techniques. Anonymity can enable censorship resistance and freedom of behavior without fear of persecution. Anonymity is mostly used to hide user identity. If user identity is hidden, access control cannot be deployed. On the other hand, if anonymous communication channels are used, a channel listener is not able to understand the messages sent on the channel or who has sent it.

A way to provide anonymity is to give users fake identities. Fake identities can be ensured by smartcard techniques where the real identity of the user is

only known by the authority which distributes the smartcards. In this case, the authority must be considered as a TTP. In **Past**, smartcards are used to allow users to obtain necessary credentials to join the system in an anonymous fashion.

Systems like **Office SharePoint Workspace** organize users into anonymous groups called workspaces. Users are known within their workspace and they are anonymous for users in other workspaces. This choice can be explained by the fact that users do not need to be anonymous to their friends or to co-workers who are authenticated in order to access their workspace.

In **OneSwarm**, anonymity is only used to protect a user identity from a third-party monitoring. Users in OneSwarm perform their queries by using anonymous routes. However, a server has the complete knowledge of the query initiator identity, so access control is possible.

Dagster, Freenet, and Tangler maintain privacy by using anonymous communication.

In **Dagster**, an anonymous channel between owners or requesters and servers is created by using Anonymizer[13], a trustworthy third party (TTP). Instead of requesting web data directly, a user sends the request to Anonymizer that forwards the request appropriately. The content is then delivered to Anonymizer that returns it to the requesting user. Anonymizer can only be used to retrieve data content and the user is required to trust that it will not reveal her identity and the requested data content.

In **Freenet**, rather than moving directly from sender to recipient, messages travel through peer to peer chains, in which each link is individually encrypted, until the message finally reaches its recipient. Each peer knows only about its immediate neighbors, so the end points could be anywhere in the network. Not even the peer immediately after the sender can tell whether its predecessor was the message's originator or was merely forwarding a message from another peer. Similarly, the peer immediately before the receiver cannot tell whether its successor is the final recipient or will continue to forward it.

In **Tangler**, privacy is maintained by using anonymous communication but also identities. Tangler ensures that a user can retrieve data without revealing their identity. Users publish documents by anonymously submitting blocks to servers. Servers can communicate with each other both directly and anonymously (by using other servers as a mixed network [6]).

In **SwarmScreen**, privacy is maintained by using random connections. They propose a privacy-preserving layer for P2P systems that disrupts community identification by obfuscating users' network behavior. Users can achieve plausible deniability by simply adding some percentage (between 25 and 50 %) of additional random connections that are statistically indistinguishable from natural ones.

[13] Anonymizer is an online service that attempts to make activity on the Internet untraceable. It accesses the Internet on the user's behalf, protecting personal information by hiding the source identifying information. `http://www.anonymizer.com/`.

Trust techniques. Trust techniques are used in P2P systems in order to reduce the probability of data privacy violation. The right to access data can be given to peers who are trustworthy and forbidden to peers who are untrustworthy.

Mainly, P2P systems that preserve privacy in distributed data storage do not trust data servers. The potential malicious behavior of peers that store data (i.e., servers) can be prevented with cryptography techniques. The systems analyzed here use trust techniques to verify trustworthiness of peers who want to access data stored on servers.

In **Past**, server peers trust owner peers thanks to a smartcard held by each peer wanting to publish data in the system. Smartcards are given by TTPs called brokers who are fully trusted by owner and server peers. A smartcard ensures the integrity of identifiers and trustworthiness assignment of the user which held it. Without a TTP, it is difficult to prevent attackers from misbehaving in the system.

On the other hand, in systems that provide massive data sharing, data owners are usually considered trustworthy and cryptography techniques are used to prevent the malicious behavior of servers. These systems are thus interested in verifying trustworthiness of requesters who want to access private data.

In **Office SharePoint Workspace**, peers can determine how much can they trust other peers through their authentication status. A peer A can optionally organize its contacts by how they were authenticated or check their authentication status by the color of their name. The names of directly authenticated contacts, which are trustworthy, are displayed in green. Other contacts in A's workspace, which are also trustworthy, are displayed in teal. Contacts in other workspaces trusted by the A's domain administrator are displayed in blue. Contacts that are not authenticated are displayed in black, and duplicated names that conflict are displayed in red. The color of the name can be used in the verification of trustworthiness of the requester. Requesters who have their name in black or red are considered untrustworthy and thus they may not gain data access.

In **OneSwarm**, data are located and transferred through a mesh of untrusted and trusted peers populated from user social networks. Peers explicitly define a trust level for a persistent set of peers. This requires some notion of identity to allow peers to relate real-world trust relationships to overlay connections. Public keys can be used as identities in order to verify trustworthiness of the peers. These public keys can be exchanged in three ways. First, requesters discover and exchange keys with owners over the local area network. Second, peers can rely on existing social networks, e.g., Google Talk or Facebook, to distribute public keys. Third, peers can email invitations to friends. Invitations include a one-time use capability that authenticates the recipient during an initial connection, during which public key exchange occurs. OneSwarm also supports key management within a group. It allows peers to subscribe to one or more community servers. A community server maintains a list of registered peers and can delegate trust regarding a subset of their peers.

Cryptography techniques. Cryptography is largely used by P2P systems in order to protect private data from unauthorized access. Encryption techniques are used to prevent malicious servers from reading private data, while digital checksums are used to detect if malicious peers are modifying or corrupting private data.

Usually, symmetric-key encryption is used to protect data content. Symmetric key generation is less expensive than asymmetric key generation. Since a large number of keys is needed to encrypt data, P2P systems have found more interest in symmetric-key encryption.

In **OceanStore**, data are encrypted using symmetric keys. Encryption keys are distributed to users who are allowed to access data.

In **Piazza**, published data are encrypted with symmetric keys in order to restrict peers from accessing data in accordance with the owners' preferences.

In **Freenet**, all data are encrypted with symmetric keys before publication. This is done mainly for political or legal reasons where servers might wish to ignore the content of the data stored. Data encryption keys are not included in network messages. Owners distribute them directly to end users[14] at the same time as the corresponding data identifiers. Thus, servers cannot read their own files, but users can decrypt them after retrieval.

In **Office SharePoint Workspace**, data are encrypted on the communication channels. Data that may be temporarily stored on servers are also encrypted using symmetric keys kept by owners, thus preventing potentially malicious servers from reading data. However, a user has the choice to delegate her identity management to servers hosted by Microsoft[15] or a TTP. If so, this one will have access to the encryption keys.

Other systems like Mnemosyne, Dagster, and Tangler use block encryption.

In **Mnemosyne**, data are divided into blocks. In order to store data, each block is encrypted using the cryptographic hash function SHA-256 and the Advanced Encryption Standar (AES), and written to a pseudo-randomly chosen location. With a good enough cipher code and key, the encrypted blocks will be indistinguishable from the random substrate, so an attacker cannot even identify the data. On the other hand, users who have the data name and key can reconstruct the pseudo-random sequence, retrieve the encrypted blocks, and decrypt them.

In **Tangler**, data are broken into a number of small blocks (shares). Each of these blocks is treated independently and stored on a subset of the participating servers. Blocks are then entangled with other random blocks, which obscures the real content of the block and makes it unreadable. In order to reconstruct a data block, users have to retrieve a minimum number of blocks of the appropriate

[14] Freenet does not use access control techniques thus key distribution is not restricted.

[15] Microsoft kept their right to collect some information about use of the Office SharePoint Workspace software and other activities "outside" of workspaces, as explained in their privacy statement at `http://office.microsoft.com/en-us/help/privacy-supplement-for-microsoft-office-groove-2007-HA010085213.aspx`

shares. By simply stripping away the random value, users can find the original data block.

In **Dagster**, data are separated into blocks, then the user generates a symmetric key for each block. Each block is encrypted with the corresponding key and sent to servers using anonymous channels. In order to reconstruct data, a number of blocks is needed along with the decryption keys.

Other systems, such as **OneSwarm**, combine public key encryption with symmetric encryption. While symmetric keys are used to encrypt data, public keys are used to share symmetric keys in a secure manner.

Having private data encrypted prevents unauthorized peers from reading their content. This contributes to protect data content privacy, although it does not protect data from being corrupted or deleted. To protect data from suppression and corruption, cryptographic hash functions (digital checksums) can be used.

In **OceanStore**, the data are named using a secure hash over the data content, giving them globally unique checksums. This provides data integrity, by ensuring that requested data have not been corrupted, since the checksum of corrupted data will be different than the globally unique checksum.

In **Freenet**, when a user publishes data which she later intends to update, she first generates a public-private key pair and signs the data with the private key. Data are published under a pseudo-unique binary key (i.e., hash key), but instead of using the hash of the data contents, the data identifier itself is used (a signature-verifying key). Signature-verifying keys can be used to verify that data content has not been tampered with.

In **Past**, the user smartcard generates reclaim certificates, containing the data identifiers and included in the user request. When processing a request, the smartcard of a server peer first verifies that the signature in the reclaim certificate matches the one in the data certificate stored with the data. This prevents unauthorized users from reclaiming the ownership of data.

To summarize our evaluation, we can see that depending on the target application, the majority of the compared P2P systems guarantee:

- Protection against unauthorized reads, by using data encryption.
- Protection against corruption and deletion, by using data checksums.
- Limited disclosure, by using access control and trust techniques.
- Anonymity and denial of linkability, by using anonymity techniques.
- Content deniability, by using data encryption.

However, these P2P systems do not support purposes. Purpose specification is essential for privacy protection as recommended by the OECD guidelines, and should be taken into account in the complete data management cycle.

3 Hippocratic databases

Inspired by the Hippocratic oath and its tenet of preserving privacy, Hippocratic databases (HDB) aim at incorporating privacy protection within relational database systems [2]. The important concepts of HDBs are the following.

- Privacy policies. A privacy policy defines for each column, row, or cell of a table (a) the usage purpose(s), (b) the external recipients, and (c) the retention period.
- Privacy authorizations. A privacy authorization defines which purposes each user is authorized to use on which data.

HDBs define ten founding principles to protect data privacy according to users preferences.

1. Purpose Specification. For personal information stored in the database, the purposes, for which the information has been collected, shall be associated with that information.
2. Consent. The purposes associated with personal information shall have the consent of the owner of the personal information.
3. Limited Collection. The personal information collected shall be limited to the minimum necessary for accomplishing the specified purposes.
4. Limited Use. The database shall run only those queries that are consistent with the purposes for which the information has been collected.
5. Limited Disclosure. The personal information stored in the database shall not be communicated outside the database for purposes other than those for which there is a consent of the owner of the information.
6. Limited Retention. Personal information shall be retained only as long as necessary for the fulfillment of the purposes for which it has been collected.
7. Accuracy. Personal information stored in the database shall be accurate and up-to-date.
8. Safety. Personal information shall be protected by security safeguards against theft and other missappropriations.
9. Openness. An owner shall be able to access all information about her stored in the database.
10. Compliance. An owner shall be able to verify compliance with the above principles. Similarly, the database shall be able to address a challenge concerning compliance.

In an HDB, queries are submitted along with their intended purpose. Query execution preserves privacy by using query rewriting and restrictions by column, row, or cell.

Purpose specification. Purpose specification is the cornerstone of an HDB. It states that purposes should be specified and attached to data items to control their usage. In order to do this, simple specification language such as Platform for Privacy Preferences (P3P) [9] can be used as a starting point. P3P is a

standard developed by the World Wide Web Consortium. Its goal is to enable users to gain more control over the use of their personal information on web sites they visit. P3P provides a way for a Web site to encode its data-collection practices in a machine-readable XML format, known as a P3P policy [9], which can be programmatically compared against a user's privacy preferences [23]. In [20,21] authors propose ideas for reducing the complexity of the policy language which include arranging purposes in a hierarchy. Subsumption relationships may also be defined for retention periods and recipients.

Limited disclosure. Limited disclosure is another vital component of an HDB system. It states that the private data shall not be disclosed for purposes other than those defined by the data owner. A scalable architecture for enforcing limited disclosure rules and conditions at the database level is proposed in [25]. For enforcing privacy policies in data disclosure, privacy policies can be stored and managed in the database. These policies are expressed in high-level privacy specification languages (e.g., P3P). Enforcing privacy policies does not require any modification to existing database applications. Authors provide techniques for enforcing a broad class of privacy policies by automatically modifying all queries that access the database in a way that the desired disclosure semantics is ensured. They examine several implementation issues, including privacy metadata storage, query modification algorithms, and structures for storing conditions and individual choices.

HDB implementation. Subsequent works have proposed solutions for implementing HDBs. In [1], authors address the problem of how current relational DBMS can be transformed into their privacy-preserving equivalents. From specifications of privacy policies, they propose an algorithm that defines restrictions (on columns, rows, and cells) to limit data access. In [4], authors propose query modification techniques and access control to ensure data privacy based on purposes. They propose to organize purposes in a tree hierarchy where the root is the most general purpose and the leafs the more specific ones. In this way, if data access is allowed for a purpose x, all descendant purposes of x are also allowed. They also propose data labeling (with allowed purposes) at different granularity levels (table, column, row, or cell). In addition, they propose some SQL modifications to include purposes, for instance **Select** column-name **From** table-name **For** purpose-name.

HDBs are the first privacy techniques that include the notion of purpose in relational databases. They are essential to users who would like to know for which purpose their data are used. However, enforcing HDBs in P2P systems is a challenge which we address in the next sections.

4 PriMod

P2P systems analyzed in Section 2 can not be used in our illustrative example on the medical OLSN, (introduced in Section 1) where participants are allowed to

use medical data depending on the purpose specification made by data owners. Thus, a new privacy solution for P2P-based OLSNs is necessary that:

- Allows to specify access purposes by using HDB principles.
- Limits data disclosure by using purpose-based access control.
- Protects data against unauthorized reads.
- Allows to detect corruption or unauthorized deletion.

PriMod [15,18,17], a data privacy model for P2P systems, is proposed to answer the need of data owners to share their sensitive data in a P2P system while preserving data privacy. It makes no assumptions about the P2P system organization. The unique important hypothesis is that each peer has a unique identifier in the system for all its connections[16]. Figure 1 shows how the purpose notion is mainspring of PriMod functionalities. PriMod allows owners to specify their privacy policies (PPs) and to publish data for specific purposes and operations. They can choose between publishing only their data references (e.g., filenames, primary keys, etc.) or publishing encrypted data content. Requesters can search for sensitive data but must specify the access purpose and operation in their requests, thus they are committed to their intended and expressed use of data. Requesters can also ask which sensitive data they can access for a particular purpose.

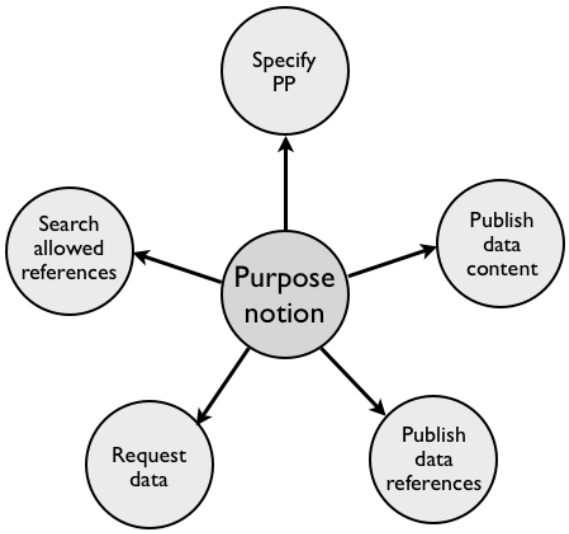


Fig. 1. Purpose as mainspring of PriMod

[16] In [5,35], authors have treated peer identification. We are fully aware of the impact of identification on user privacy. However, peer identities do not necessarily reveal users real identities thus user privacy can be somehow protected.

To summarize, the PriMod assets are the following:

- It benefits from P2P assets in data publishing and sharing while offering data privacy protection based on access purposes.
- It can be easily integrated to any P2P system.
- It proposes/uses privacy policies concepts and defines models for trust and data management.
- It offers operations for publishing data content, publishing references, requesting, and purpose-based searching.
- Data owners can define their privacy preferences in privacy policies.
- Sensitive data are associated with privacy policies. This association creates private data ready to be published in the system.
- Requests are always made for particular purposes and operations.
- Trust techniques are used to verify trustworthiness of requester peers.

In the following, Section 4.1 presents the privacy policy model of PriMod. Section 4.2 presents the data model. Section 4.3 introduces the functions of PriMod.

4.1 Privacy policy model

In PriMod, each data owner should define her privacy preferences. Those privacy preferences are registered in PPs independently of data. Once defined, they are attached to appropriate data. PPs are dynamic because they can vary with time. For instance, at the end of the medical treatment, a doctor will only allow reading access to other doctors for analyzing the patient medical record. Updating diagnosis will not be allowed anymore. We consider that each owner is responsible for defining and maintaining her PPs in an independent way.

Inspired from the Platform for Privacy Preferences (P3P) [9], Figure 2 shows a PP model. This model does not claim to be exhaustive, but shows information about PPs that can include:

Authorized Users. It is a list of users who are authorized to access data, a kind of ACL. A user can be an individual or a group.

Purpose. An access purpose states the data access objective. With this concept, an owner is able to specify the purposes for which its data can be accessed by users.

Operations. An operation determines what a peer can do with data. We use three basic operations, read, write, and disclose, but others can be defined.

- **Read.** A peer can read the data content.
- **Write.** A peer can modify the data content with the following operations: insert, update, delete.
- **Disclose.** A peer is able to disclose shared data for other peers. Disclosure can be limited or unlimited. If a peer disclosure right is limited, it cannot give disclosure rights on the data to other peers.

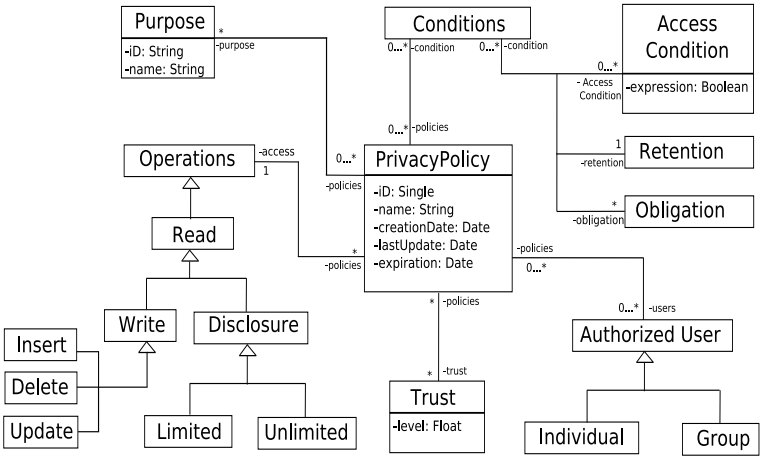


Fig. 2. Privacy policy (PP) model

Conditions. Conditions state the access conditions a user should respect, the obligations a user should accomplish after accessing data, and the limited time for data retention.

- **Access condition.** Conditions state under which semantic condition data can be accessed. This may concern data values, for example age>10.
- **Obligation.** Obligations state the obligation a user must accomplish after the data access. For example, researcher Ri should return research results after using the patient records.
- **Retention time.** The retention time states the time limit of retention of the data. For example, the local copy obtained by a requester of a patient record should be destroyed after 1 year of use.

Minimal trust level. It is the minimal trust level a requester peer should have in order to gain access to data.

4.2 Data model

In order to respect PPs, they should be associated with data. In the following, we use relational tables, however any type of data can be considered (files, XML documents, rich text files, etc.).

Data table. Each owner peer stores locally the data it wants to share (see Table 5). Those data can be stored in relational tables called *data tables*. The unique restriction about data tables is that primary keys should be generic and impersonal to respect privacy and to not disclose any information. If considered data are files, their identifiers or names should be impersonal.

Data table DTj							
Id (PK)	**SS**	**Name**	**Country**	**Birthdate**	**Gender**	**Smoker**	**Diagnosis**
Pat1	001044001001	Alex	France	2000	Male	No	NO cardiovascular disease
Pat2	900344001001	Bea	France	1990	Female	No	NO cardiovascular disease

Table 5. Data table of doctor Dj

Privacy policy table. Data contained in PPs are stored in a table named *privacy policies table* (see Table 6). To simplify, all elements of Figure 2 are not included. In this table, one tuple corresponds to one PP. The same PP can be used with different data. Each policy contains operations (read, write, or disclose), allowed users, access purposes, conditions (if they exist), and the required minimal trust level of allowed users.

Privacy policy table PPTj					
Id (PK)	**Operation**	**User**	**Purpose**	**Condition**	**Minimal trust level**
PP1	Read	Pharmacists, Doctors	Consulting record	Birthdate < 2000	0.5
PP2	Read	Researchers	Researching on cardiovascular disease	—	0.6

Table 6. Privacy policy table of doctor Dj

Purpose table. Information about the available purposes are stored in a table named *purpose table*. A tuple of the purpose table contains the purpose identifier, the purpose name, and the purpose description. We recall that in HDB, purposes can be organized in a hierarchy. To simplify, in PriMod, we make abstraction of such hierarchy.

Trust table. Each peer maintains a local *trust table* that contains the trust level of some peers in the system. A tuple of the trust table contains the identifier of a peer, its trust level, and a cell defining if this peer is consider as a *friend* or not locally.

Private data table. This table joins data to privacy policies. It allows fine-grained access control by specifying which table, column, line, or cell can be accessed by preserving which privacy policy. For instance, in PD1 of Table 7, only some columns of the data table DTj (those who do not disclose patients identities) are concerned by the privacy policy PP1 where pharmacists and doctors can read records of patients who were born before 2000.

Private data table j				
Id (PK)	**Data**			**Privacy**
	Table	**Column**	**Id**	**Policy**
PD1	DTj	Birthdate, Gender, Smoker, Diagnoses	—	PP1
PD2	DTj	Country, Birthdate, Gender, Smoker, Diagnosis	—	PP2

Table 7. Private data table of doctor Dj.

Purpose-based data reference table. To ease data searching, a purpose-based index is necessary. Information about the references of data allowed for particular purposes and operations for particular requesters are stored in a table named *purpose-based data reference table* (PBDRT for short). This purpose-based index allows requesters to know which data they can accessed for a particular purpose and operation. Each tuple of this table is identified by a *key*, obtained for instance by hashing the couple $(purpose, operation)$ (see Table 8). Besides the key, a tuple contains the identifiers of requesters and the list of data references the are allowed to access.

Purpose-based data reference table j (PBDRT j)		
Key (PK)	**requesterID**	**DataRefList**
hash(diagnosis, write)	Doctor1	{DataRef1}
	Doctor2	{DataRef1, DataRef2}
hash(research, read)	Doctor1	{DataRef1}
	Scientist1	{DataRef1}
	Scientist2	{DataRef1, DataRef3, DataRef5}
hash(accessing, read)	Patient1	{DataRef1}
	Patient2	{DataRef2}
	Patient3	{DataRef3, DataRef5}

Table 8. Purpose-based data reference table (PBDRT) of doctor Dj.

4.3 PriMod functions

PriMod proposes the next set of functions.

Publishing data. PriMod provides two ways of publishing sensitive data indicating the PP that users should respect. An owner may choose to publish her data content or only data references. In the first case, data storage is protected from malicious servers by using cryptography techniques. In the second case, there is no need of data encryption since references do not show any private information about the data if they are well-chosen (i.e., personal information such as security numbers and addresses should not be used in references).

Boolean publishData(data, PPId). Owner peers use this function to publish data content in the system. The second parameter is the privacy policy that dictates the usage conditions and access restrictions of the published data. This function returns true if data content is successfully distributed, false otherwise. It is similar to a traditional publishing function. To protect data privacy against potential malicious servers, before distribution, data content is encrypted (by using symmetric cryptography) and digital checksums are used to verify data integrity. Symmetric keys are stored locally by the owner. Requesters must contact owners to retrieve keys and decrypt requested data.

Boolean publishReference(data, PPId). Owner peers use this function to publish data references in the system while data content are stored locally. This function returns true if data references are successfully distributed, false otherwise. Servers store data references and help requesters to find data owners to obtain data content. Publishing only data references allows owners to publish private data while being sure that data content will be provided to right requesters. This hypothesis can not be guaranteed in the previous function because malicious servers may misbehave by returning encrypted data to unauthorized peers.

Requesting data. For requesters, how data have been published is transparent and a unique function to request data is proposed by PriMod.

Data request(dataRef, purpose, operation). Requester peers use this function to request data (*dataRef*) for a specific purpose (e.g., researching, diagnosis, or analysis) to perform a specific operation (i.e., read, write, or disclose). This function returns the requested data if the requester has corresponding rights, otherwise it returns null. This function compels requesters to specify the access purposes and the operations that they have the intention to apply to requested data.

This explicit request is the cornerstone of this work, it commits requesters to use data only for the specified purposes and to perform only specified operations. Legally, this commitment may be used against malicious requesters if it is turned out that obtained data have been used differently.

P2P Model	Guaranteed properties				
	Protection against unauthorized reads	Protection against corruption and deletion	Limited disclosure		Content deniability
			Owners	Requesters	
PriMod	Yes, due to encryption	Yes, due to data digital checksums		Yes, due to access control	Yes, due to encryption
	Privacy Techniques				
	Access control	Anonymity	Trust techniques	Data encryption	Data integrity protection
	Use of purpose-based access control		Use of trust levels	Symmetric encryption	Use of content hashing as a checksum

Table 9. PriMod: used privacy techniques and guaranteed properties

TrustLevel searchTrustLevel(requesterID). Owner peers use this function to search the trust level of the requester *requesterID*. This function returns the trust level of the requester if it is found else it returns null. This trust level is used in the requesting process to verify the trustworthiness of the requester in order to give him access rights.

Purpose-based reference searching. PriMod provides users with a function for purpose-based reference searching based on the PBDRT. This function allows requesters to know which data they are authorized to request for a particular purpose and operation. This prevents users from denying knowledge about their access rights. The allowed data reference lists (contained in the PBDRT) can be created transparently while publishing data. These lists can be published periodically in the system.

DataRefList dataRefSearch(purpose, operation). Requester peers use this function to know the data they are authorized to access for a specific purpose and operation. This function returns a list of data references of data the requester is authorized to acces (*dataRefList*). If the list is empty, the function returns null. This request by peer protects privacy because it avoids that all users know which are the access rights of other users and know the complete list of available data (global schema).

4.4 Analysis

Table 9 compares PriMod to works shown in Section 2.

- Private data in PriMod are protected against malicious reads, corruption, and deletion by using encryption and data checksums.
- Data disclosure is limited by using access control only for requesters.

– Unlike all the models presented, the notion of purpose is omnipresent in PriMod.
– PriMod does not guarantee anonymity since it is not designed to protect user behavior but only data privacy.
– Trust levels are used to prevent malicious behavior of requesters.

5 PriServ

PriServ is a privacy service that implements PriMod. Figure 3 shows the Priserv's architecture, which is on top of a DHT layer. This DHT layer has two functional components: one is in charge of the routing mechanism that supports the *lookup()* function as well as the dynamicity of peers (join/leave of peers); the other ensures key-based data searching and data distribution by implementing the *put()* and *get()* functions. These two layers provide an abstraction from the DHT.

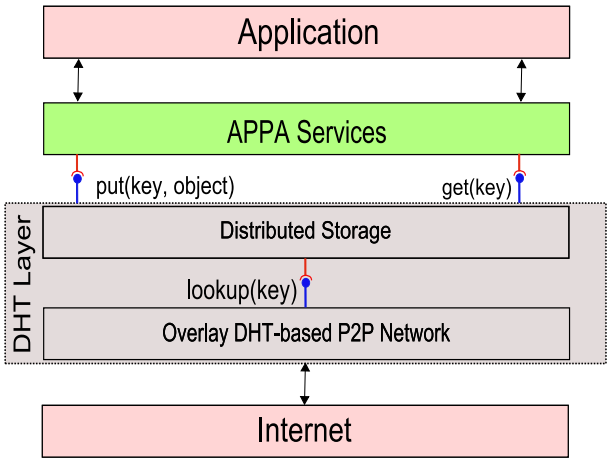


Fig. 3. Global architecture

Conceptually, PriServ is an APPA (Atlas Peer-to-Peer Architecture) service [3]. APPA is a data management system for large-scale P2P and Grid applications. The PriServ implementation uses Chord for its efficiency and simplicity, however, any DHT can be used. PriServ uses the traditional get() and put() functions of DHTs to locate and publish data, each incurring $O(logN)$ messages.

– $put(key, data)$ stores a key k and its associated data object in the DHT.
– $get(key)$ retrieves the data object associated with k in the DHT.

Data keys in PriServ are created by hashing the triplet (*dataRef, purpose, operation*). We consider that *dataRef* is a unique data reference, *purpose* is

the data access purpose and *operation* is the operation that can be executed on requested data with respect to the corresponding privacy policy. Thus, the same data with different access purposes and different operations have different keys.

5.1 PriServ architecture

Figure 4 shows the component-based architecture of PriServ. It provides five interfaces to the application layer: publishReference(), publishData(), request(), dataRefSearch(), and dataRefPut(). The first four correspond to PriMod operations. The last one allows users to construct a distributed purpose-based index used in the dataRefSearch() function. PriServ also provides two retrieve functions necessary for the interaction between requesters and owners. As said before, it uses the two traditional interfaces of the DHT layer (put() and get()).

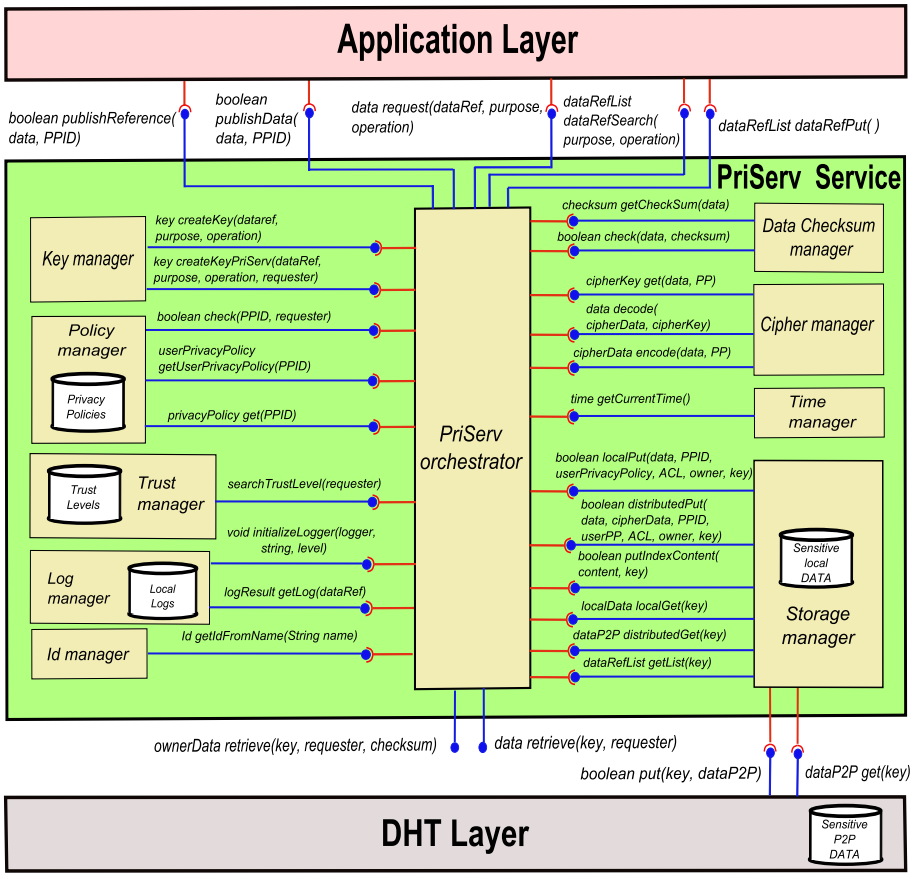


Fig. 4. PriServ architecture

Inside PriServ, several components are gathered around an orchestrator which organizes PriServ activities. The orchestrator may execute three different workflows depending on the role of the peer (owner, requester, or server/storer).

Storage manager. Its role is to manage data storage. Data storage is made locally before data are stored in the P2P system. To store data in the P2P system, this component invokes the put() and get() functions of the DHT layer.

Policy manager. Its role is to manage owner PPs. Data owners organize their privacy preferences in PPs. From PPs, this component creates "user privacy policies" that should be respected by requesters. From PPs, this component extracts the list of authorized users. This list is a kind of access control list (ACL) that contains only allowed users. To simplify, we consider that users may belong to groups, so, this ACL contains mainly a list of groups of users and maybe some individual users known in advance.

Trust manager. Its role is to manage trust levels. We consider that each peer stores locally a list of peers and their corresponding trust levels. If a required trust level does not exist locally, the trust manager asks for it to other peers. In PriServ, three ways of obtaining trust levels are used, namely, with-friends, without-friends, and with-or-without-friends. Section 5.2 explains those algorithms.

Cipher manager. Its role is to manage cryptography in PriServ. It offers a function that creates a symmetric cipher key for each pair (data, PP). It also offers two functions to encrypt and decrypt data. In this work, symmetric-key algorithms can be used because they are generally much less computationally intensive than other cryptography algorithms. However, PriServ is independent of the encryption technique used.

Data checksum manager. Its role is to check the integrity of data. This component offers a function to calculate digital data checksums (e.g., by using MD5). Digital checksums can be used by owners to verify if servers have tampered with the data. PriServ is independent of the techniques used to create data checksums.

Key manager. Its role is to generate data keys used in the put() and get() functions. It creates data keys by hashing the triplet *(data references, purposes, operations)*. This component offers two functions, the first one, used during the put() process, returns the created data key and the second, used during the get() process, returns the created data key and adds it the identifier of the requester.

Id manager. Its role is to identify peers from their names (e.g., URI). Peer identifiers are used in access control to authorize or prohibit access to data.

Log manager. Its role is to manage logs. It stores logs in a dedicated database on each peer. These logs can be used for recovery and auditing process.

Time Manager. Its role is to give the current time. It is used for synchronizing clocks of all connected peers.

PriServ orchestrator. It is the central component of PriServ. According to the role of the peer, the orchestrator executes a different workflow by using the components introduced before.

- ***Owner orchestrator.*** Its role is to orchestrate the owner functionalities. It is responsible for publishing in the P2P system references or data depending on the called function (publishData() or publishReference()). It is also responsible for directly returning data or symmetric keys during the requesting process (retrieve()). It interacts with the application layer for publishing and with the requester orchestrator for retrieving.
- ***Requester orchestrator.*** Its role is to orchestrate data requesting. It interacts with the application layer for requesting and with the owner orchestrator for retrieving.
- ***Server orchestrator.*** Its role is to orchestrate the server functionalities. For that, it interacts with the DHT layer to store data for the P2P system and to return stored data.

5.2 PriServ mean functions

PriServ implements the PriMod functions so it offers to the application layer two ways for publishing and allows searching data and data references for a particular purpose and operation. The main procedures are publishReference(), publishData(), request(), dataRefSearch(), dataRefPut(), and searchTrustLevel(). All but the last function use the DHT organization. The searchTrustLevel() function uses instead an unstructured P2P approach as we will see latter.

In the following, consider 6 peers with identifiers P23[17], P25, P31, P33, P51, and P60. Consider also that:

- P23 is an owner peer.
- P31 and P25 are requester peers.
- P33, P51, and P60 are server peers.

Boolean publishReference(data, PPId). Owners use this function to publish data references under a particular PP. Publishing only data references and storing data locally allow owners to provide themselves their data to right requesters.

When the owner orchestrator receives the publishReference() call from the application, it uses the algorithm shown in Figure 5 to publish data references in the P2P system and to store data locally. The object sent to the P2P system contains the *conditions* that data requesters should respect (userPP) when using the data (see Figure 2), the ACL that servers should verify, and the owner id of the data reference.

[17] To distinguish data keys from peer keys, we prefix peer keys with letter P.

```
Owner orchestrator
0:   publishReference(data, PPID)
1:   begin
2:     privacyPolicy = policyManager.get(PPID);
3:     userPP = policyManager.getUserPrivacyPolicy(PPID);
4:     key = keyManager.createKey(data.dataref, privacyPolicy.purpose,
                                  privacyPolicy.operation);
5:     storageManager.localPut(data, PPID, userPP, privacyPolicy.ACL,
                               owner, key);
6:     key2 = keyManager.createKey(null, privacyPolicy.purpose,
                                   privacyPolicy.operation);
7:     storageManager.addRef(key2, data.getDataref(), privacyPolicy.ACL);
8:   end;

Owner storage manager
9:   localPut(data, PPID, userPP, ACL, owner, key)
10:   begin
11:     dataTable.localSave(data);
12:     privateDataTable.localSave(data.dataRef, PPID);
13:     dataP2P = createDataP2P(userPP, ACL, owner);
14:     DHT.put(key,dataP2P);
15:   end;
```

Fig. 5. Algorithm of the publishReference() function

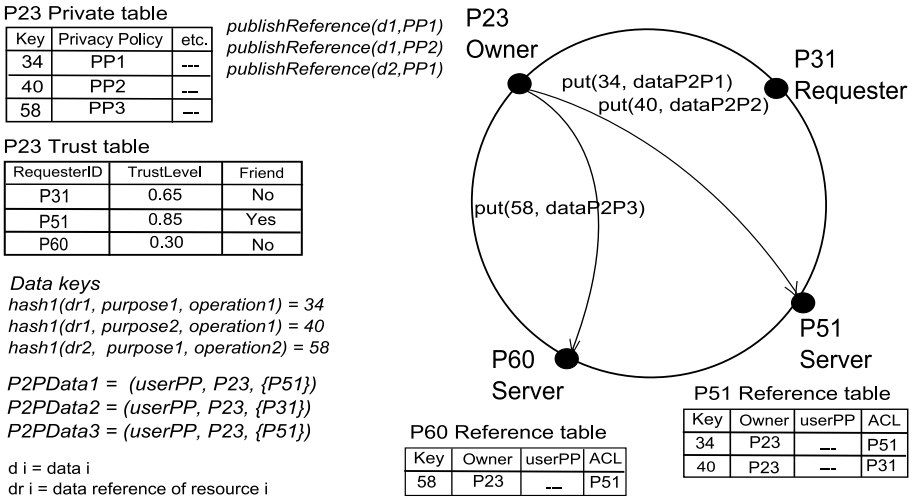


Fig. 6. Example of the publishReference() function

Figure 6 illustrates this algorithm. P23 shares data with keys 34, 40, and 58. *hash*1 is used to produce these keys from data identifiers (IDri), purposes, and operations. From the DHT principle, P51 is the server peer responsible for key 34 and 40 and P60 for 58. Only data references of P23 are published with its identifier by using the put function.

Boolean publishData(data, PPId). Owners use this function to publish data content under a particular PP (PPId). To protect data privacy against potential untrusted servers, before distribution, data content is encrypted (by using symmetric cryptography), and digital checksums are used to protect data integrity from servers.

When the owner orchestrator receives the publishData() call from the application, it uses the algorithm shown in Figure 7 to publish the data content in the P2P system and to store locally a copy of the data.

```
Owner orchestrator
0:    publishData(data, PPID)
1:    begin
2:        privacyPolicy = policyManager.get(PPID);
3:        userPP = policyManager.getUserPrivacyPolicy(PPID);
4:        key = keyManager.createKey(data.dataref, privacyPolicy.purpose,
                                privacyPolicy.operation);
5:        cipherData = cipherManager.encode(data, privacyPolicy);
6:        storageManager.distributedPut(data, cipherData, PPID,
                            userPP, privacyPolicy.ACL, owner, key);
7:        key2 = keyManager.createKey(null, privacyPolicy.purpose,
                                        privacyPolicy.operation);
8:        storageManager.addRef(key2, data.getDataref(), privacyPolicy.ACL);
9:    end;

Owner storage Manager
10:   distributedPut(data, cipherData, PPID, userPP, ACL, owner, key);
11:   begin
12:       dataTable.localSave(data);
13:       privateDataTable.localSave(data.dataRef, PPID);
14:       dataP2P = createDataP2P(cipherData, userPP, ACL, owner);
15:       DHT.put(key, dataP2P);
16:   end;
```

Fig. 7. Algorithm of the publishData() function

Figure 8 illustrates this algorithm. P23 shares data with keys 36, 42, and 59. *hash*1 is used to produce keys from data identifiers (IDri), purposes, and operations. P51 is the server peer responsible for key 36 and 42, and P60 for key 59. P2P data are created by P23 by encapsulating encrypted data, the P23 identifier, and the corresponding ACL that contains requesters' identifiers allowed to access this data. Then, P2P data are published by using the put function.

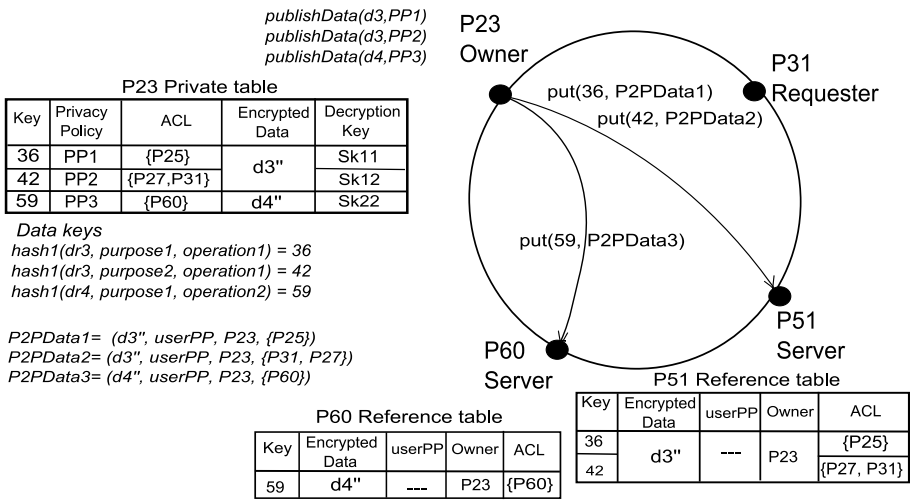


Fig. 8. Example of the publishData() function

Data request(dataRef, purpose, operation). Requesters use this function to request data (*dataRef*) for a specific *purpose* to perform a specific *operation*. This function compels requesters to specify the access purposes and the operation that they will apply to requested data. When a requester orchestrator receives the request() call from the application, it uses the algorithm shown in Figure 9. For requesters, the way data have been published is transparent (publishData or publishReference), so they always use this request function.

Figure 10 illustrates the data requesting algorithm on the example where encrypted data are published (see Figure 8). P31 requests data for purpose2 and operation1 that corresponds to key 42, so the storage manager does a $get(42)$. The peer which identifier is equal to or follows 42 is P51. P51 returns P23 which is the owner peer of 42 and the encrypted data corresponding to 42. In this example, we consider that P51 misbehaves and returns a corrupted data $cr'3$. P31 calculates a checksum of the received data, then it contacts directly P23 to retrieve the decryption key corresponding to 42 ($retrieve(42, P31, checksum)$). P23 verifies the information contained in the privacy policy of 42 (PP2). In this example, consider that the trust table of P23 contains the trust level of P31 that we suppose is 0.7. As the trust level of P31 is higher than the level required in PP2 that is 0.6 (see Table 6), the access is granted. P23 calculates the checksum of the data corresponding to 42 and compares it to the checksum value sent by P31. P23 finds out that the checksums are not equal and deduces that someone has misbehaved. Since P31 has a corrupted data, P23 sends the encrypted data with the encryption key.

During the request process, the servers do an access control based on the ACL sent by the owner during the publishing process. The data owner is always contacted by the requester either to request the data content (if only references have been published) or the decryption key (when encrypted data content have been

```
Requester orchestrator
0:   data request(dataRef, purpose, operation)
1:   begin
2:      data = null;
3:      key = keyManager.createKeyPriServ(dataRef, purpose,
                                          operation, requester);
4:      dataP2P = storageManager.distributedGet(key);
5:      if (dataP2P contains cipher data) then
6:         checksum = dataCheckSumManager.getCheckSum(dataP2P.cipherData);
7:         ownerData = dataP2P.owner.retrieve(key, requester, checksum);
8:         if (ownerData contains a cipherKey) then
9:            data = cipherManager.decode(dataP2P.cipherData,
                                          ownerData.cipherKey);
10:        else
11:          if (ownerData contains data) then
12:            data = ownerData.data;
13:          end if;
14:        end if;
15:     end if;
16:     if (dataP2P contains a data reference) then
17:      data = dataP2P.Owner.retrieve(key, requester);
18:     end if;
19:     return data;
20:  end;

Requester storage manager
21:  dataP2P distributedGet(key)
22:  begin
23:     dataP2P = DHT.get(key);
24:     return dataP2P;
25:  end;
```

Fig. 9. Algorithm of the request() function

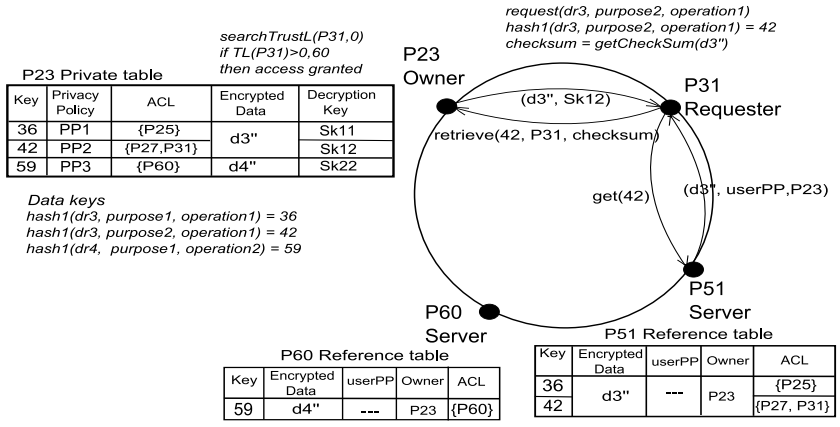


Fig. 10. Example of the request() function

published). Before retrieving data, owners check the trust level of requesters as you will see in the function searchTrustLevel().

To use this request function, it is necessary to know the references of available data in the system. This information is maintained in an index that can be centralized or distributed (see Section 2). A centralized index represents a point of failure and potential bottlenecks. Besides, it implies to trust one single peer (or server) that has the control and the responsibility of maintaining this important meta-information. We argue that when preserving privacy, the distribution of control is essential, which is why PriServ implements a distributed index.

DataRefList dataRefSearch(purpose, operation). PriServ implements the purpose-based reference searching function of PriMod. The index represented by PBDRT (see Table 8) in PriServ is implemented in a distributed way by using the DHT organization. The couple $(purpose, operation)$ is hashed to create the keys of the index. Keys are assigned to peers that are responsible of maintaining information about the peers that can request data for the purpose and operation represented by the key. The hash function used to produce these keys may be different from the one used to publish data. Thus, each peer maintains a PBDRT of all the keys for which its id is the closest in the DHT organization, which gives a partial view of the global index.

During the publishing process, the key for the purpose and operation is created and a new data reference is added to the PBDRT index (a) when publishing references (lines 6-7 of Figure 5) and (b) when publishing encrypted data (lines 7-8 of Figure 7). The addRef() function used by owners during publishing follows the algorithm of Figure 11.

```
Owner storage manager
0:   addRef(key, dataRef, ACL)
1:   begin
2:      dataP2P=createDataP2P(dataRef, ACL);
3:      DHT.put(key, dataP2P);
4:   end;
```

Fig. 11. Algorithm of the addRef() function

When the requester orchestrator receives the dataRefSearch() call from the application, it uses the algorithm shown in Figure 12. The requester orchestrator asks the key manager to create a key by hashing the purpose and the operation. Then, it asks its storage manager to obtain the data references it has access rights for the specified purpose and operation from the P2P system. The storage manager gets the data reference list corresponding to the key by invoking the get() function of the DHT.

Figure 13 illustrates the dataRefSearch() function. P25 requests data references it has right access for *purpose1* and *read* that corresponds to key 37

```
Requester orchestrator
0:   dataRefList dataRefSearch(purpose, operation)
1:   begin
2:      dataRefList = null;
3:      key = keyManager.createKeyPriServ(null, purpose, operation,
                                           requesterID);
4:      dataRefList = storageManager.getList(key);
5:      return dataRefList;
6:   end;

Requester storage manager
7:   dataRefList getList(key)
8:   begin
9:      dataRefList = DHT.get(key);
10:     return dataRefList;
11:  end;
```

Fig. 12. Algorithm of the dataRefSearch() function

(*get*(37)). The peer whose identifier is equal to or follows 37 is P51. P51 returns the data reference list *{dr1, dr8}* corresponding to key 37 and requester P25. You can see that P25 can request dr1 for which owner is P23 but also dr8 for which owner is not shown in the figure.

To optimize the update of the index, a periodic publication of references can be done by using the dataRefPut() algorithm shown in Figure 14. The idea is that, if an owner publishes many data for the same (*purpose*, *operation*) only one update is done. For that, a local PBDRT is maintained and *flushed* periodically. With this periodical update, lines 6-7 of Figure 5 (resp. lines 7-8 of Figure 7) should be suppressed. See the local PBDRT of P23 in Figure 13.

The last function that PriServ implements focuses on searching the trust level of requesters. PriServ uses trust levels to make the final decision of sharing or not data. The trust level reflects a peer reputation wrt other peers. A peer can have different trust levels at different peers. The peer reputation influences its trust level. Peers which are suspicious have lower trust level than peers considered as honest. A peer can have locally the trust levels of some well known peers or peers it has interacted with. If a peer P does not have a particular trust level it can ask for it to its *friends*. A friend is a peer considered as honest from the point of view of P and the number of friends can vary from one peer to another.

The implementation of the searchTrustLevel() function does not use the DHT organization, but uses an unstructured overlay. Thus, this function has been redefined to take into account a Time To Live (TTL).

TrustLevel searchTrustLevel(requesterID, nestedLevel). Trust levels are considered in the range [0..1]. A peer with a trust level of 1 is completly trustworthy. A peer with a trust level of 0 has very bad reputation. During requesting, if the trust manager of the data owner has the trust level of the requester it does

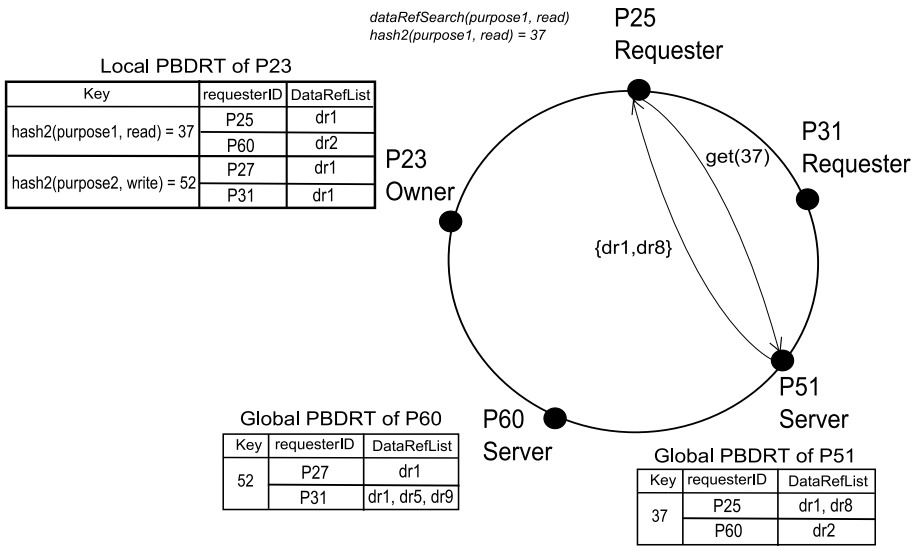


Fig. 13. Example of the dataRefSearch() function

```
Owner orchestrator
0:   dataRefPut()
1:   begin
2:      for each key i contained in the localPBDRT do
3:          contentKey = localPBDRT.getContent(key);
4:          storageManager.putIndexContent(contentKey, key);
5:      end for;
6:   end;

Owner storage manager
7:   putIndexContent(contentKey, key)
8:   begin
9:       dataP2P=createDataP2P(contentKey);
10:     DHT.put(key, dataP2P);
11:  end;
```

Fig. 14. Algorithm of the dataRefPut() function

not have to contact other peers. Otherwise, PriServ defines three methods for searching the requester trust level. Choosing one of them depends on the number of friends of the owner. Briefly, the three methods are explained below, and only the algorithm of the first one is presented. More details can be found in [16].

```
0:  trustLevel searchTrustLevel(requesterID, nestedLevel)
1:  begin
2:     requesterTrustLevel = 0;
3:     if (nestedLevel has reached maxDepth) then
4:       if (trustLevel of requesterID in trustTable) then
5:         requesterTrustLevel=trustLevel of requesterID;
6:       else
7:          return -1;
8:       end if;
9:     else
10:       if (trustLevel of requesterID in trustTable) then
11:         requesterTrustLevel=trustLevel of requesterID;
12:       else
13:          nestedLevel is incremented;
14:          nbPeersContacted = 0;
15:          for each friend do
16:            FTL = trustLevel of friendID;
17:            RTL = friendTrustManager.searchTrustLevel(requesterID,
                                                      nestedLevel);
18:            if (RTL != -1) then
19:                FTL*RTL is added to requesterTrustLevel;
20:                nbPeersContacted is incremented;
21:            end if;
22:          end for;
23:          requesterTrustLevel = requesterTrustLevel/
                                  nbPeersContacted;
24:       end if;
25:    end if;
26:    return requesterTrustLevel;
27:  end;
```

Fig. 15. Algorithm of the searchTrustLevel() function: with-friends version

With-friends algorithm. This version of the searchTrustLevel function considers that each peer has at least one friend (Figure 15). With this assumption, the data owner asks its friends for the trust level of the requester. Each received trust level (RTL) is weighted with the trust level (FTL) of the sending friend. The final trust level is computed from the received trust levels as the average,

the maximum, the minimum, etc. This searching is recursive. If a friend does not have the requested trust level, it asks for it to its friends and the number of nested levels (*nestedLevel*) is incremented. Recursion is limited to *maxDepth*. The maximum number of contacted friends can also be limited to a predefined number.

Without-friends algorithm. In this algorithm, we consider that peers have no friends. In this case, data owners ask for the trust level of the requester to the subset of known peers from the DHT, i.e., their finger table .

With-or-without-friends algorithm. Here, peers may have friends or not and priority is given to ask for trust levels to friends. If a data owner has some friends, it asks them for the trust level by using the with-friends algorithm, else it asks the peers in its finger table by using the without-friends algorithm.

5.3 PriServ validation

We validated PriServ in three steps, with costs analysis, simulations, and implementation of a Java prototype. The costs analysis and results of simulation with SimJava are presented next, the prototype is presented in Section 6.

Publishing costs. Publishing data in the system conserves the logarithmic cost of the traditional put function. By using the DHT, $O(logN)$ messages are needed to publish each key. In PriServ, the number of keys is equal to the number of entries (*ept*) of the private data table. Additional costs induced by the cipher key generation and the data encryption are negligible wrt. the network costs. Thus, the publishing cost is:

$$C_{Publish} = \sum_{i=1}^{ept} O(logN) = O(ept * logN)$$

The maximum value of *ept* is equal to the number of shared data (*nbData*) multiplied by the number of purposes (*nbPurpose*) multiplied by the number of operation (*nbOperation*). At worst, each data item is shared for all purposes and all operations:

$$CMax_{Publish} = O(nbData * nbPurpose * nbOperation * logN)$$

We can see that the number of purposes and operations affects the publishing cost. Previous studies have shown that considering ten purposes allows to cover a large number of applications [30,24]. Used with ten purposes (by data item) and three operations (read, write, and disclosure), PriServ incurs a small overhead. Overall, the publishing cost remains logarithmic.

Requesting costs. Concerning the requesting cost, it is the addition of two costs: get() cost and the retrieving cost. We disregard access control, checksum calculation, and decryption costs, which are negligible wrt. network costs. The get() cost is in $O(logN)$ and the server returns its answer in one message. For data retrieval, a requester needs one additional message to contact the data owner that answers in another message.

To summarize, the requesting cost is:

$$\begin{aligned} C_{Requesting} &= C_{DHTGet} + C_{Retrieving} \\ &= O(logN) + 1 + 2 \\ &= O(logN) + 3 \\ &= O(logN) \end{aligned}$$

Trust level searching cost. The trust level searching cost (C_{STL}) depends on the trust searching algorithm:

- *With-friends algorithm.* In this case, the owner sends a message to each of its friends that in turn do the same in a nested search. This cost depends on the number of friends (NF) and the maximum depth of the nested search (MaxDepth).

$$C_{STL_{WF}} = \sum_{i=1}^{MaxDepth} NF^{\,i} = O(NF^{MaxDepth})$$

- *Without-friends algorithm.* In this case, the owner sends a message to each of the peers in its finger table, which in turn do the same in a nested search. This cost depends on the number of fingers, which is log N, and the maximum depth of the nested search (MaxDepth).

$$C_{STL_{WOF}} = \sum_{i=1}^{MaxDepth} (logN)^{\,i} = O((logN)^{MaxDepth})$$

- *With-or-without-friends algorithm.* In this case, if the owner has friends, it sends a message to each of its friends. Otherwise, it sends a message to each of the peers in its finger table. A peer contacted by an owner does the same in a nested search. The trust level searching cost depends on the number of friends (NF), the number of fingers, which is log N, and the maximum depth of the nested search (MaxDepth).

$$C_{STL_{WWF}} = O((max(logN, NF))^{MaxDepth})$$

The trust level searching cost C_{STL} can be one of the three costs $C_{STL_{WF}}$, $C_{STL_{WOF}}$, or $C_{STL_{WWF}}$. Note that if NF > log N, $C_{STL_{WWF}}$ is equal to $C_{STL_{WF}}$, else it is equal to $C_{STL_{WOF}}$. In all cases $C_{STL_{WWF}}$ can be used for C_{STL}:

$$C_{STL} = O((max(logN, NF))^{MaxDepth})$$

To summarize,

$$C_{publishing} = O(nbData * nbPurpose * nbOperation * logN)$$

$$C_{Requesting} = C_{Request} + C_{Retrieve} = O(logN)$$

$$C_{STL} = O((max(logN, NF))^{MaxDepth})$$

For the simulation, we used SimJava [13] and the Chord protocol was simulated with some modifications in the put() and get() functions. Tests consider N peers, peer keys are selected randomly between 0 and 2^n. N is set to 11, which corresponds to 2^{11} peers. This number of peers is enought to simulate collaborative applications like the medical one. MaxDepth is set to 11 and the number of friends is set to 2.

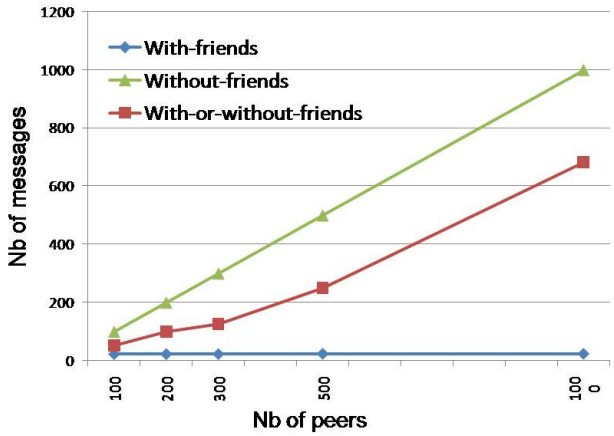


Fig. 16. Comparison of the three algorithms of trust level searching

Trust level searching introduces a large overhead because of flooding in the unstructured network. Figure 16 compares the three algorithms seen above. The with-friends case introduces the smallest cost while the without-friends case introduces the highest cost. However, intuitively, the probability to find the trust level is higher in the without-friends algorithm than in the with-friends algorithm. This is due to the fact that the number of contacted peers is higher in the without-friends algorithm, which increases the probability to find the trust level. We estimate that the with-or-without-friends algorithm is the most optimized because it is a tradeoff between the probability to find the requester trust level and the trust level searching cost.

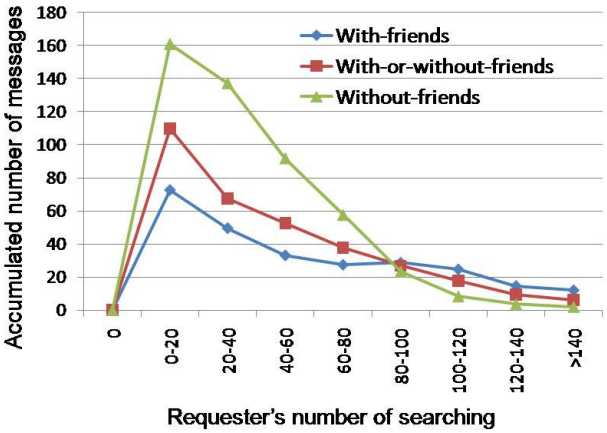


Fig. 17. Stabilization of the cost of trust level searching

Figure 17 shows that the trust level searching cost decreases with the number of requests and stabilizes. When peers ask for a trust level, answers are returned in the requesting order and the trust tables are updated with the missing trust level. Thus, the trust tables evolve with the number of searches. After a while, these tables stabilize. Thus, the number of messages for searching trust levels is reduced to a stable value. This value is not null because of the dynamicity of peers. Simulations consider that the number of peers joining the system is equal to those leaving the system. Thus, there are always new peers which do not know the requester trust level. We also observe in the figure that the trust level searching cost in the without-friends algorithm stabilizes first. This is due to the fact that a larger number of peers are contacted. The with-or-without-friends algorithm comes in second place, and the with-friends algorithm comes last. As can be seen in the comparison of the three algorithms, we find again that the with-or-without-friends algorithm is the most optimized because it is a tradeoff between the time to stabilization and the trust level searching cost.

6 PriServ prototype

A prototype of PriServ for privacy-preserving data sharing applications for online communities was developed [19]. The prototype uses the Java language, SCA (Service Component Architecture) tools[18], and RMI (Remote Method Invocation). Figure 18 shows the owner peer implementation with SCA.

[18] `http://www.oasis-opencsa.org/sca`
`http://www.obeo.fr/pages/sca/`

PriServ was tested and validated by PeerUnit[19] on Grid5000[20]. Grid5000 is a scientific instrument for the study of large scale parallel and distributed systems. The tests were done on a population of 180 peers on 42 Grid5000 nodes. Several results have validated the performance of the prototype concerning:

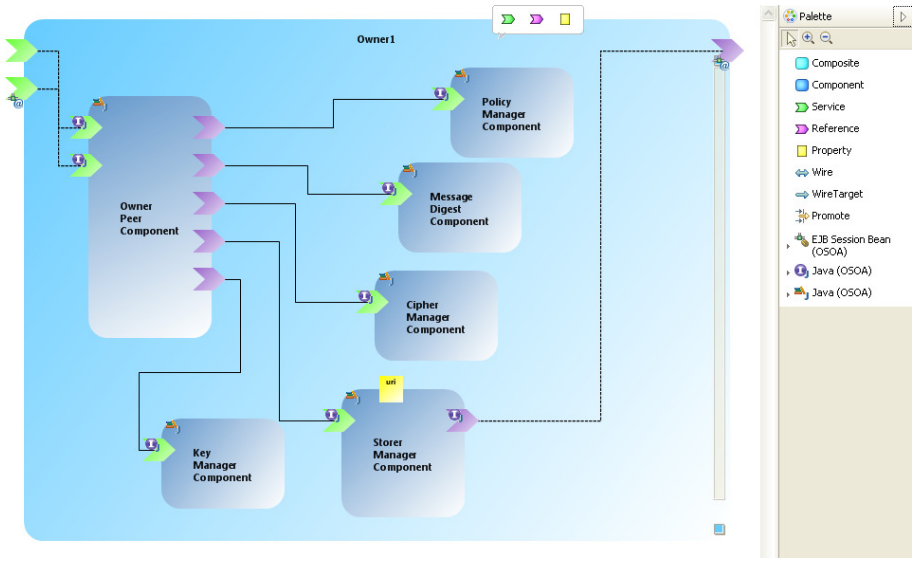


Fig. 18. Owner peer modeled with SCA

- Respect of privacy policies of owners. PriServ limits data access if a requester does not intend to respect the owner privacy policy. The test results show the failure of the request each time the access purpose or operation are different from the intended purpose and operation specified while publishing data.
- Limited access only for authorized requester. The test results show that 100 % of queries from an unauthorized requesters are systematically rejected by servers.
- Traceability of the data distribution. The test results show that it is possible to review, automatically, all the requesters who had access to data, and to check the list of those who have tried to access them. This allows the data owner to be sure that its data privacy requirements are respected.

6.1 Medical PPA

Privacy-Preserving Applications (PPA) manage sensitive data (financial projects, unpublished research results, patients records, etc.). PriServ is illustrated with

[19] `http://peerunit.gforge.inria.fr/`
[20] `http://www.grid5000.fr/`

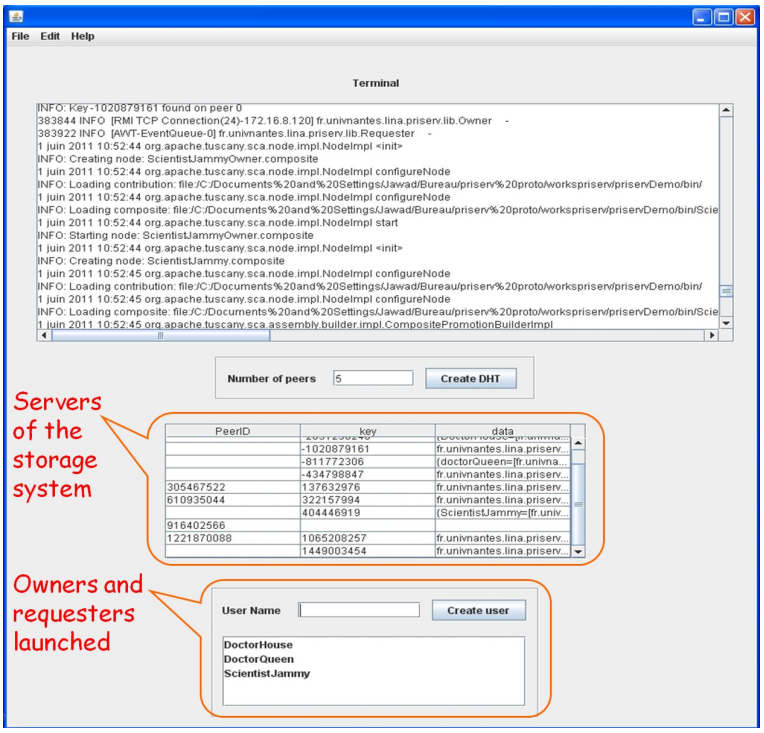


Fig. 19. Dashboard of the storage system (DHT)

the collaborative medical research application of Section 1. The participants of this applicaiton are scientists, doctors, and patients. In order to control disclosure of sensitive information (e.g., medical records owned by doctors, research results owned by scientists) without violating privacy, data access should respect the privacy preferences of their owners. In this medical PPA:

- Patients and doctors who own and manage private medical records can be considered as owners. Scientists who may use medical records for scientific research can be considered as requesters. Servers are peers of the storage system.
- Doctors may define the privacy preferences of patients in privacy policies and attach them to their medical records. For instance, a doctor may allow writing access on her information to scientists for adding comments on her diagnosis.
- Scientists may define their own privacy preferences and attach them to their research results. For instance, a scientist may allow reading access on her results to doctors for giving diagnosis.

PriServ is used for these scenarios:

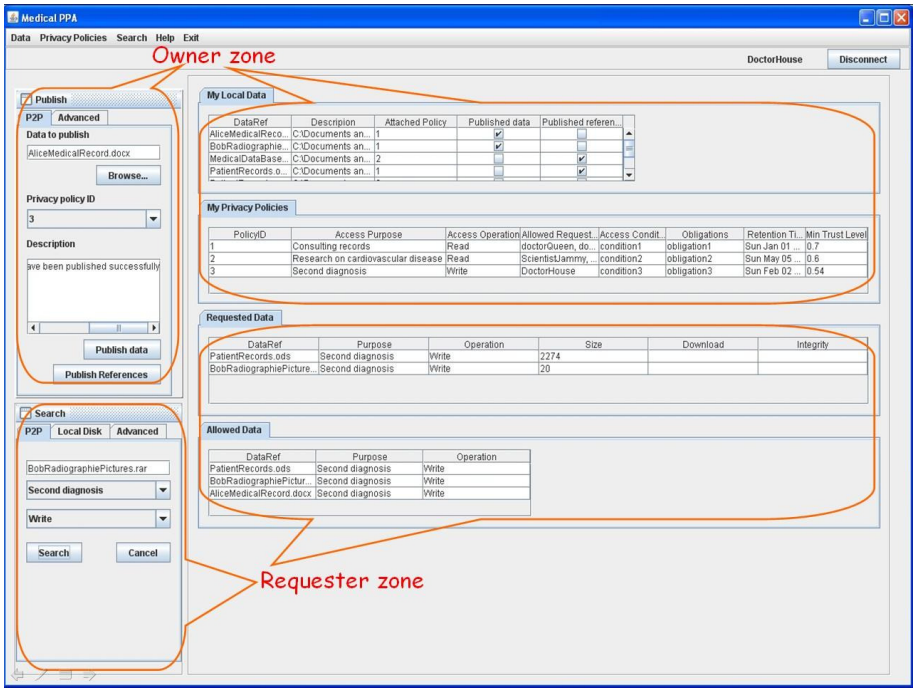


Fig. 20. GUI of the medical PPA that uses PriServ

– After defining their privacy preferences on their medical records, doctors and patients can publish their data in the system while preserving their privacy.
– Scientists can search for data by using procedures that respect the privacy preferences of the data owners.

Through a GUI, we show scenarios that exhibit important aspects of private data management: privacy policy management, data publishing, data searching, and data reference searching.

Figure 19 shows the interface through which the DHT is launched. Once launched, the DHT creates server peers whose number can be specified. Then the main user creates owners and requesters by specifying their names and clicking on *create user*. The interface of each user newly created will appear automatically. For instance, Figure 20 shows the user interface of DoctorHouse.

Scenarios. The key features of the prototype, are demonstrated through the following scenarios of the medical PPA:

Privacy policies management. This scenario is used to show how DoctorHouse (an owner) can specify his privacy preferences by defining his own privacy policies (see Figure 21). He is also able to attach different policies to a datum in order to

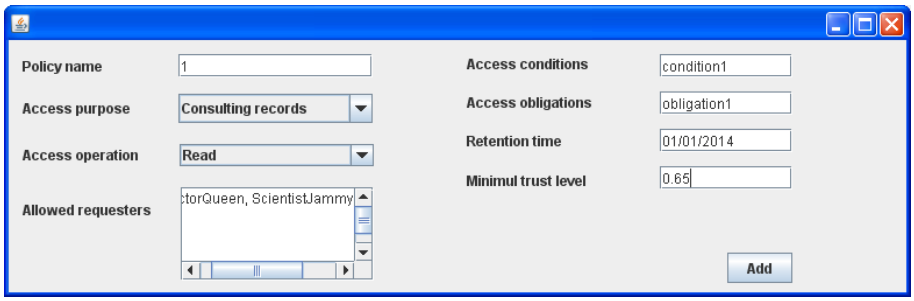
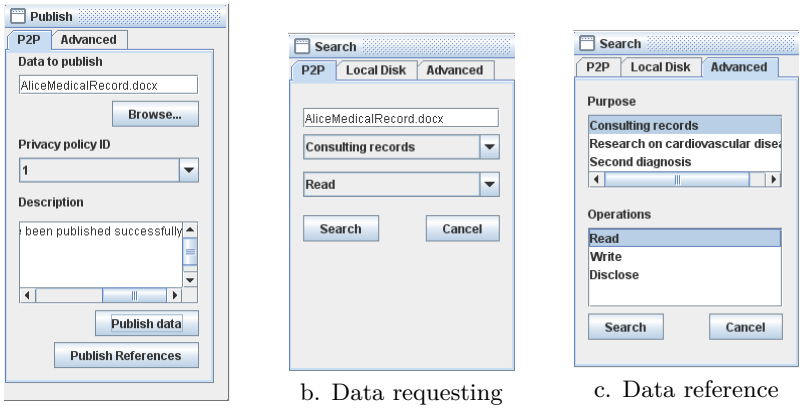


Fig. 21. Privacy policy specification interface

control the access to his data. He also has an interface that shows information about his privacy policies.



a. Data publishing interface

b. Data requesting interface

c. Data reference searching interface

Fig. 22. Publishing and requesting interfaces

Data publishing. This scenario shows how DoctorHouse can publish sensitive data in the P2P system (see Figure 22.a). He can specify for which privacy policy his data will be published. He also has the choice between publishing encrypted data or data references. He can also have a view of his published data and the policies attached to them.

Data searching. This scenario shows how ScientistJammy can search for data (see Figure 22.b). He has a choice between: (a) a local search on his own local

data, (b) a P2P data requesting. Searching is made only if he selects the purpose and the operation for the requested data.

Data reference searching. This scenario shows how ScientistJammy can search for data references for particular purpose and operation (see Figure 22.c). For this, we show that he will be able to have a list of references for data which he can access for particular purpose and operation. Then, he can access data content by specifying the data reference that he has gotten, the purpose and the operation in the data searching interface.

The PriServ web site is `https://sites.google.com/site/gddlina/priserv` and the prototype code can be found at `http://sourceforge.net/projects/priserv/`.

7 Conclusion

This chapter gave an overview of current solutions for supporting data privacy in P2P systems. Our evaluation of existing solutions is made based on the techniques used to protect privacy of data and users (i.e., access control, anonymity, trust, and cryptography) and the guaranteed privacy properties (i.e., protection against unauthorized reads, corruption and deletion, limited disclosure, anonymity guarantees, denial of linkability, and content deniability). This analysis showed that while the notion of purpose of Hippocratic databases (HDB) is gaining more attention, in particular, because of the OECD guidelines, it has not been used in P2P systems.

Then we developed in more details a complete solution (Primod and Priserv) for data privacy in P2P systems that supports the notion of purpose of HDB. PriMod, a privacy model for P2P systems, integrates the purposes notion as mainspring. Purposes are omnipresent in several process of sensitive data management. Data owners specify, through personal privacy policies, the access purpose for their data. Data publication attaches the allowed access purposes. Data requesters specify the access purpose in their requests, thus they are committed to their intended and expressed use of data.

PriServ is a privacy service that implements PriMod. The PriServ prototype combines purpose and operation-based access control, trust techniques, cryptography techniques, and digital checksums. A privacy-preserving data sharing application for online social networks illustrates this approach.

Several improvements can be made to PriMod and PriServ. The purpose-based index should be anonymized to avoid servers to know the partial view of the index they store. A more semantically rich query language may also be proposed. But above all, auditing solutions should be proposed to verify compliance of data use with the specified privacy preferences. This is still an open and challenging issue.

References

1. Agrawal, R., Bird, P., Grandison, T., Kiernan, J., Logan, S., Rjaibi, W.: Extending Relational Database Systems to Automatically Enforce Privacy Policies. In: IEEE Conference on Data Engineering (ICDE). Tokyo, Japan (April 2005)
2. Agrawal, R., Kiernan, J., Srikant, R., Xu, Y.: Hippocratic Databases. In: Very Large Databases (VLDB). Hong Kong, China (August 2002)
3. Akbarinia, R., Martins, V., Pacitti, E., Valduriez, P.: Design and Implementation of APPA. Global Data Management (Eds. R. Baldoni, G. Cortese, F. Davide), IOS Press (2006)
4. Byun, J.W., Li:, N.: Purpose Based Access Control for Privacy Protection in Relational Database Systems. Very Large Databases (VLDB) Journal 17(4) (2008)
5. Castro, M., Druschel, P., Ganesh, A., Rowstron, A., Wallach, D.S.: Secure Routing for Structured Peer-to-Peer Overlay Networks. In: Operating Systems Design and Implementation (OSDI). Boston, Massachusetts (December 2002)
6. Chaum, D.L.: Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. Communications of the ACM 24(2) (1981)
7. Choffnes, D.R., Duch, J., Malmgren, D., Guierma, R., Bustamante, F.E., Amaral, L.: SwarmScreen: Privacy Through Plausible Deniability in P2P Systems. Tech. rep., Northwestern EECS University (March 2009)
8. Clarke, I., Miller, S.G., Hong, T.W., Sandberg, O., Wiley, B.: Protecting Free Expression Online with Freenet. IEEE Internet Computing 6(1) (2002)
9. Cranor, L., Langheinrich, M., Marchiori, M., Presler-Marshall, M., Reagle, J.: The Platform for Privacy Preferences 1.0 (P3P1.0) Specification (2002)
10. Daswani, N., Garcia-Molina, H., Yang, B.: Open Problems in Data-Sharing Peer-to-Peer Systems. In: International Conference on Database Theory (ICDT). Siena, Italy (January 2003)
11. Garton, L., Haythornthwaite, C., Wellman, B.: Studying Online Social Networks. Journal of Computer-Mediated Communication 3(1) (1997)
12. Hand, S., Roscoe, T.: Mnemosyne: Peer-to-Peer Steganographic Storage. In: International Peer To Peer Systems Workshop (IPTPS). Cambridge, MA, USA (March 2002)
13. Howell, F., McNab, R.: Simjava: a Discrete Event Simulation Library for Java. In: International Conference on Web-Based Modeling and Simulation. San Diego, CA, USA (January 1998)
14. Isdal, T., Piatek, M., Krishnamurthy, A., Anderson, T.: Privacy-Preserving P2P Data Sharing with OneSwarm. Tech. rep., University of Washington (2009)
15. Jawad, M.: Data Privacy in P2P Systems. Ph.D. thesis, Université de Nantes (June 2011)
16. Jawad, M., Serrano-Alvarado, P., Valduriez, P.: Protecting Data Privacy in Structured P2P Networks. In: Data Management in Grid and P2P Systems (Globe). Linz, Austria (September 2009)
17. Jawad, M., Serrano-Alvarado, P., Valduriez, P., Drapeau, S.: A Data Privacy Service for Structured P2P Systems. In: Mexican International Conference in Computer Science (ENC). México D.F., México (September 2009)
18. Jawad, M., Serrano-Alvarado, P., Valduriez, P., Drapeau, S.: Data Privacy in Structured P2P Systems with PriServ. In: Bases de Données Avancées (BDA). Namur, Begium (October 2009)
19. Jawad, M., Serrano-Alvarado, P., Valduriez, P., Drapeau, S.: Privacy Support for Sensitive Data Sharing in P2P Systems. In: Bases de Données Avancées (BDA), demonstration paper. Rabat, Morocco (October 2011)

20. Karjoth, G., Schunter, M., Waidner, M.: Platform for Enterprise Privacy Practices: Privacy-Enabled Management of Customer Data. In: Workshop on Privacy Enhancing Technologies. San Francisco, CA, USA (April 2002)
21. Kleinberg, J., Papadimitriou, C.H., Raghavan, P.: On the Value of Private Information. In: Theoretical Aspects of Rationality and Knowledge (TARK). Siena, Italy (June 2001)
22. Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S.E., Eaton, P.R., Geels, D., Gummadi, R., Rhea, S.C., Weatherspoon, H., Weimer, W., Wells, C., Zhao, B.Y.: OceanStore: An Architecture for Global-Scale Persistent Storage. In: Architectural Support for Programming Languages and Operating Systems (ASPLOS). Cambridge, MA, USA (November 2000)
23. Langheinrich, M.: A P3P Preference Exchange Language (APPEL1.0) Specification (2001)
24. Liberty Alliance Project, Privacy Preference Expression Languages (PPELs). http://projectliberty.org/liberty/content/download/371/2670/file/Final_PPEL_White_Paper.pdf
25. LeFevre, K., Agrawal, R., Ercegovac, V., Ramakrishnan, R., Xu, Y., DeWitt, D.J.: Limiting Disclosure in Hippocratic Databases. In: Very Large Databases (VLDB). Toronto, Canada (September 2004)
26. Mell, P., Grance, T.: The NIST Definition of Cloud Computing. National Institute of Standards and Technology 53(6) (2009)
27. Miklau, G., Suciu, D.: Controlling Access to Published Data Using Cryptography. In: Very Large Databases (VLDB). Berlin, Germany (September 2003)
28. Nejdl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmér, M., Risch, T.: Edutella: A P2P Networking Infrastructure Based on RDF. In: ACM World Wide Web Conference (WWW). Hawaii, USA (May 2002)
29. Ozsu, M.T., Valduriez, P.: Principles of Distributed Database Systems. Springer, 3rd edn. (2011)
30. 1.0 P3P Purposes of Data Collection Elements. http://p3pwriter.com/LRN_041.asp
31. Pfitzmann, A., Hansen, M.: A Terminology for Talking about Privacy by Data Minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management. Tech. rep., Dresden University of Technology (2009)
32. Roncancio, C., del Pilar Villamil, M., Labbé, C., Serrano-Alvarado, P.: Data Sharing in DHT based P2P systems. Transactions on Large Scale Data and Knowledge Centered Systems I 5740 (2009)
33. Rowstron, A., House, G.: Storage Management and Caching in PAST, a Large-scale, Persistent Peer-to-Peer Storage Utility. In: Symposium on Operating Systems Principles (SOSP). Banff, Alberta, Canada (October 2001)
34. Rowstron, A.I.T., Druschel, P.: Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In: ACM/IFIP/USENIX Middleware Conference (MIDDLEWARE). Heidelberg, Germany (November 2001)
35. Ryu, S., Butler, K., Traynor, P., McDaniel, P.: Leveraging Identity-Based Cryptography for Node ID Assignment in Structured P2P Systems. In: Advanced Information Networking and Applications Workshops (AINA). Niagara Falls, Canada (May 2007)
36. Stoica, I., Morris, R., Karger, D.R., Kaashoek, M.F., Balakrishnan, H.: Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In: ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM). San Diego, CA, USA (August 2001)

37. Stubblefield, A., S.Wallach, D.: Dagster: Censorship-Resistant Publishing without Replication. Tech. rep., Rice University (2001)
38. Tatarinov, I., Ives, Z.G., Madhavan, J., Halevy, A.Y., Suciu, D., Dalvi, N.N., Dong, X., Kadiyska, Y., Miklau, G., Mork, P.: The Piazza Peer Data Management Project. ACM Special Interest Group on Management of Data (SIGMOD) Record 32(3) (2003)
39. Waldman, M., Mazières, D.: Tangler: A Censorship-Resistant Publishing System Based on Document Entanglements. In: Computer and Communications Security (CCS). Philadelphia, PA, USA (November 2001)
40. Westin, A.F.: Privacy and Freedom. Atheneum, New York, USA (1967)
41. Zhao, B.Y., Huang, L., Stribling, J., Rhea, S.C., Joseph, A.D., Kubiatowicz, J.: Tapestry: A Resilient Global-Scale Overlay for Service Deployment. IEEE Journal on Selected Areas in Communications 22(1) (2004)